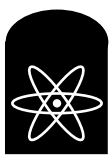
UCRL-ID-122243

Vendor Assessment and Software Plans

Prepared by
G. G. Preckshot
J. A. Scott

Prepared for
U.S. Nuclear Regulatory Commission



FESSP
Fission Energy and Systems Safety Program

Lawrence Livermore National Laboratory

Disclaimer

This work was supported by the United States Nuclear Regulatory Commission under a Memorandum of Understanding with the United States Department of Energy, and performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

UCRL-ID-122243

Vendor Assessment and Software Plans

Version 2.0

G. G. Preckshot
J. A. Scott

Manuscript Date: November 1995

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Vendor Assessment and Software Plans

1.0 Introduction

Several previous studies performed for the Nuclear Regulatory Commission by Lawrence Livermore National Laboratory have focused on characteristics of software development processes that are important for the development of high-integrity software. These include software reliability (NUREG/CR-6101, Lawrence [1993]) and software design factors (NUREG/CR-6294, Lawrence and Preckshot [1994] and Ploof and Preckshot [1993]). Ploof and Preckshot [1993] has been included as Appendix B of this report. In addition, recent analyses of standards important to the development of software for the safety systems of nuclear power plants have indicated the importance of the understanding and use of a complete framework of standards in the development of such software (Scott et. al. [1995]). Finally, Preckshot [1994] (Appendix A) addressed the assessment of software development processes used by software vendors. The latter work defined a set of steps to be followed in conducting vendor assessments. This report relates, in detail, the vendor assessment steps to the planning audits proposed in NUREG/CR-6101. The correspondence of the vendor assessment steps to the design factor categories of NUREG/CR-6294 is also discussed.

1.1 Scope and Purpose

The scope of this paper covers assessment of vendors producing software under programs that comply with NQA-1, IEEE Std 603, and IEEE Std 7-4.3.2, and that use additional standards that are compatible with the framework defined by these three standards. The intention of the paper is to describe a correspondence between software design, reliability, and vendor assessment factors uncovered by previous work and the content and execution of plans required by standards.

1.2 Design Outputs Versus Traditional Manufacturing Outputs

For software quality assurance (QA), design outputs occupy the same place as manufacturing outputs do in traditional quality assurance theory. The only manufacturing process clearly involved in software is the replication of executable code either on disk or in ROM, and quality assurance of the replication process is relatively trivial. Arguably, compilation and linking may also be a manufacturing process; however, the design outputs from the software development process form a far more significant portion of the product than the replication, compilation, or linking processes or their possible statistical variations. Vendor assessment and software development process assessment are, therefore, essential to software product evaluation.

1.3 Software Development Process

The software development process converts design inputs, or software requirements derived from system requirements, to design outputs through a design process. The software design process is described by tangibles that prescribe the design process, such as the various planning documents addressed by this paper, NUREG/CR-6101, and various standards of the standards framework described below. The software design process produces a number of intermediate design outputs, such as high-level design, detailed design, and code. The software development process consists of a series of design and design verification stages, with the design outputs of one stage being the design inputs of

the following stage. The manufacturing process consists of duplicating executable binary code at the end of several design and implementation stages, a step that is trivial compared to the complexity of the preceding stages. Therefore, the intermediate and final design outputs of a software product are of great importance. It is tempting to apply statistical methods to the products of a software design process in an attempt to characterize the design process. Unfortunately, unambiguous and repeatable process measures and an underlying theoretical basis have been elusive (Preckshot [1993], Preckshot and Scott [1994]). Consequently, the quality of a software development process is judged by heuristics and attributes, such as those described in NUREG/CR-6101, NUREG/CR-6294 , and Preckshot [1994]. Because there are no reliable process measures, the problem with heuristics becomes one of application. This paper establishes a connection between vendor assessment heuristics and the Lawrence questionnaires for various standards-prescribed software plans so that vendor assessment heuristics can be applied by reviewing vendor software planning and software plan execution.

2.0 Summary of Key Aspects of Previous Work

Previous work that is being brought together in this paper consists of a description of the nuclear quality assurance and software process standards framework extant in the United States, a proposed vendor assessment process based on design factors work, and software reliability guidance in the form of questionnaires to apply to software planning documents.

2.1 Standards Framework

Adhering to standards in the development of systems and software does not guarantee high integrity; however, it does ensure that currently accepted practices will be employed. In general, standards represent consensus thinking on the state of the practice and provide an accepted basis for the structuring of software development processes. To structure an effective software development process based on standards, it is necessary to be familiar with the detailed requirements of individual standards as well as the relationships among the standards, including software standards and standards specific to the nuclear industry. Multiple standards frameworks exist and variations of a given standards framework are likely in different software development environments. The important commonalities are the structuring of a standards-based, high-integrity software development process and a complete understanding of the role played by the suite of related standards in the quality assurance process. An incomplete understanding of the standards framework and its relationship to a proposed software development process may leave assurance objectives unfulfilled and expend effort without commensurate results.

A standards framework that is important for nuclear power plants in the U.S. includes ASME NQA-1-1994, “Quality Assurance Requirements for Nuclear Facility Applications,” IEEE nuclear industry standards such as IEEE 603-1991, “Criteria for Safety Systems for Nuclear Power Generating Stations,” and IEEE 7-4.3.2-1993, “Criteria for Digital Computers in Safety Systems of Nuclear Power Generating Stations.” In addition, various IEEE software engineering standards provide detailed guidance on software development. Figure 1 shows the relationships among 10 CFR 50 Appendices A and B, NRC Regulatory Guides, and the system-level standards in this framework. Figure 2 shows the interrelationships among the current versions of the standards referenced in Figure 2. Figure 3 shows the system elements covered by the various standards in the framework.

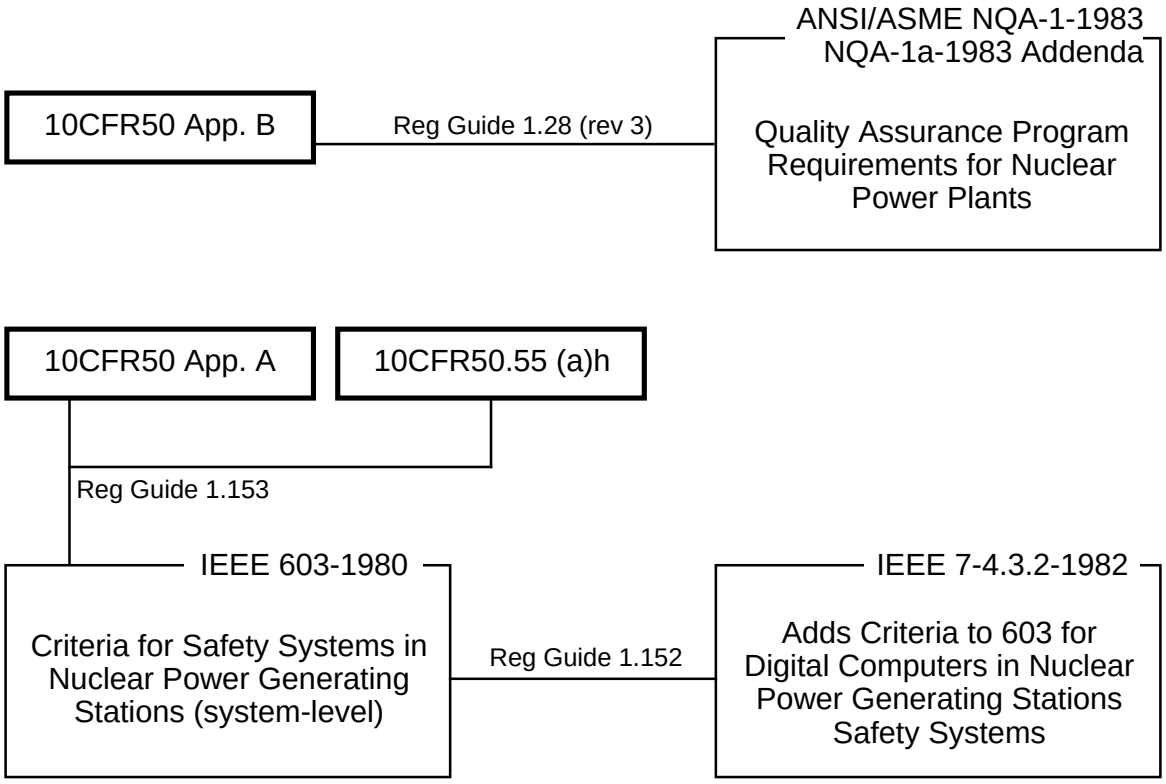


Figure 1. Key Standards and Regulatory Guide Endorsement

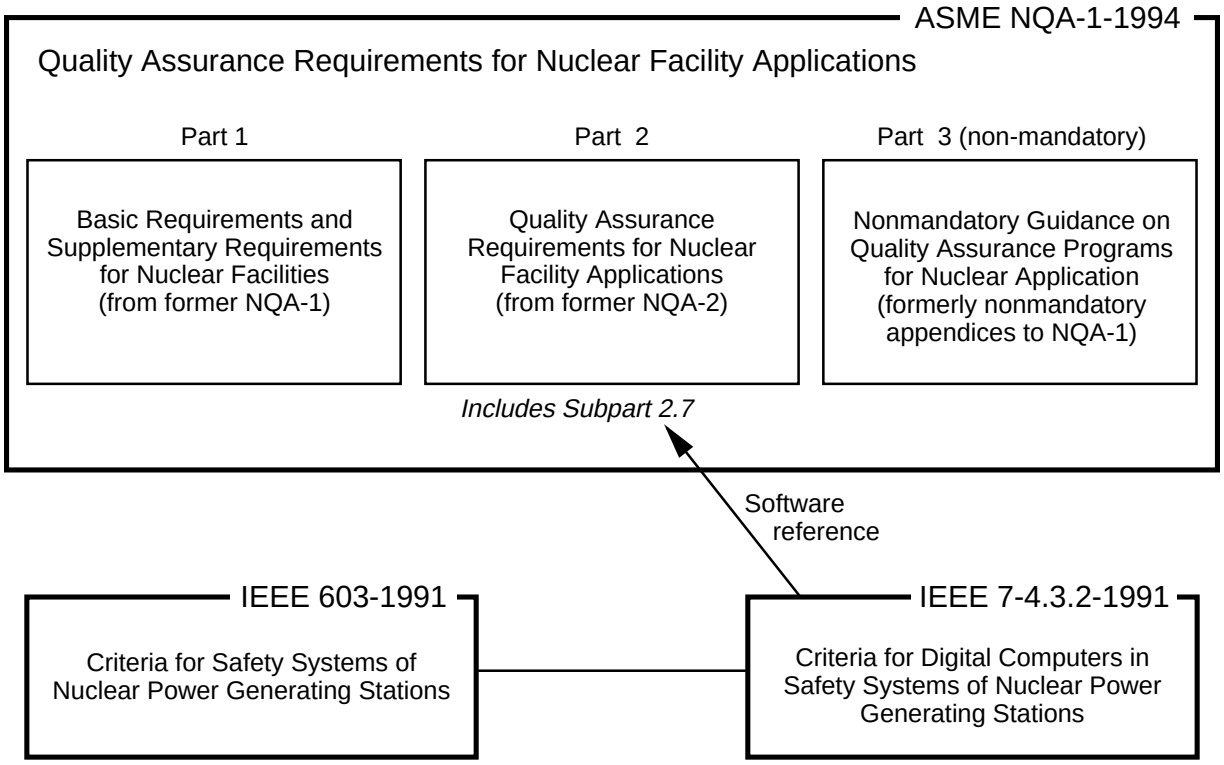


Figure 2. Relationships of Current Versions of the Key Standards

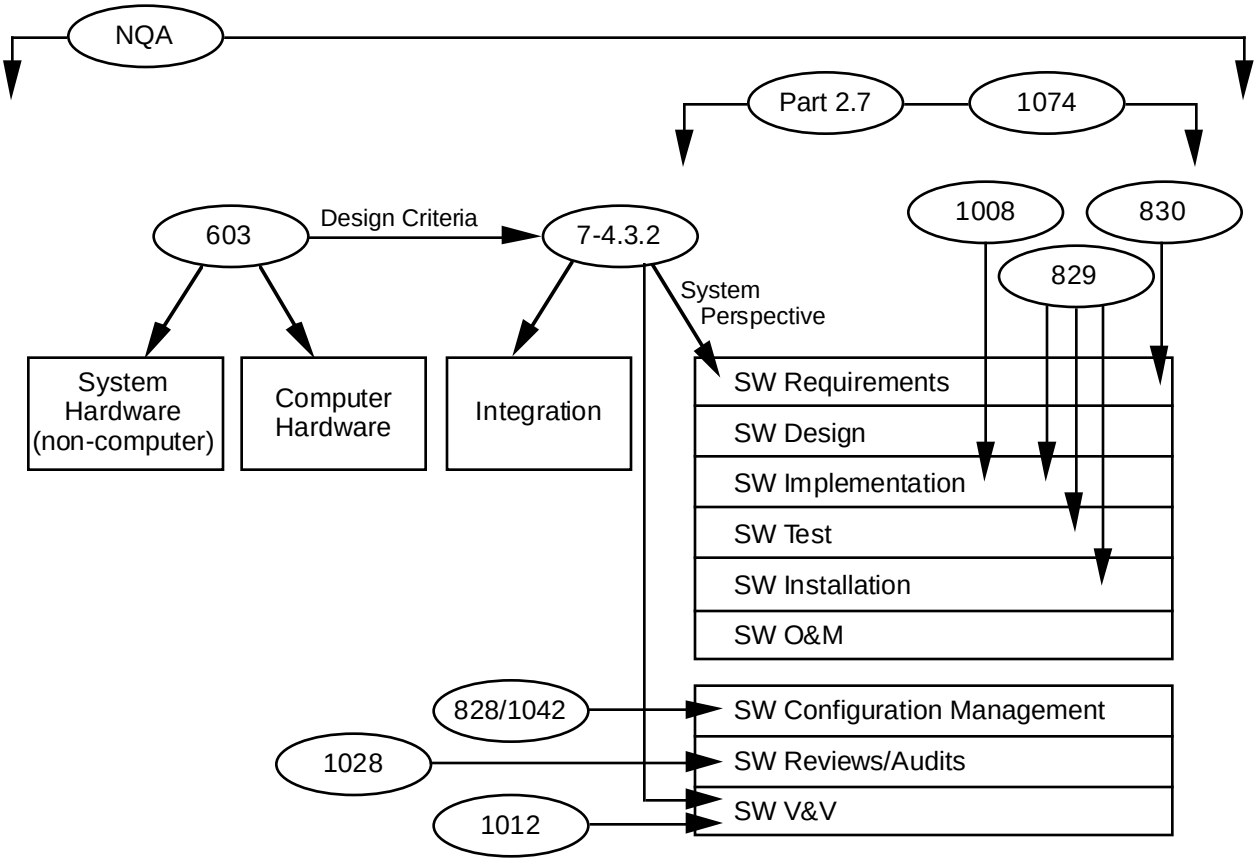


Figure 3. System Elements Addressed by Standards in the U.S. Standards Framework

2.2 Summary of Vendor Assessment Steps

The proposed vendor assessment steps are listed below for reference and in Appendix A [Preckshot 1994]. The seven major steps are shown in Table 1. These steps were formulated by considering "design factors" (described in the following section) as a guide for reviewing vendor documentation and vendor interviews. During the formulation of the vendor assessment steps, the documentation to be reviewed was not limited to planning documents, and the interviews to be done were not limited to audits of plan performance.

2.3 Summary of Software Design

The vendor assessment steps are based on interpretation of software design factors as applied to a vendor's software development process. NUREG/CR-6294 defines the term "design factor," describes 7 design factors as mandatory (no deviation is acceptable) for high-integrity software development, and 9 as essential (some deviation is acceptable). A total of 74 design factors were determined by interviews with professional software developers, a workshop involving several noted academics and technical contributors, and a review of a broad spectrum of consensus standards.

Table 1. Seven major steps to vendor assessment

1.	Determine Criticality
2.	Determine Organizational Interfaces
3.	Evaluate Vendor History
4.	Evaluate Current Program
5.	Check for Negative Factors
6.	Evaluate Commitments (or Execution of Commitments)
7.	Perform Periodic Reviews

3.0 Mapping Vendor Assessment Activities to Plans

In Section 4.0, the 7 steps of the vendor assessment process are mapped to the plan assessment questions given in NUREG/CR-6101. Both the content (which questions apply to specific areas) and timing (when to ask the questions) of the assessment process are covered. A correspondence between general guidance given in Preckshot [1994], and specific assessment guidance in NUREG/CR-6101 is made. Suggestions for evaluating vendor history, evaluating the current program, and evaluating commitments using the plan assessment questions are noted.

Plan assessment should also verify the software organization's understanding and commitment to standards or their use in company-specific standards. Plan assessment (and assessment of performance of previously reviewed plans) is made at intervals, such as those listed below:

1. At the beginning of the project
2. After requirements are done, but before design
3. At the completion of top-level design
4. At the completion of detailed design
5. At the completion of implementation
6. After system integration and validation
7. After each delivery and plant installation.

4.0 Plan Audits

Assessment questions in §4.1 of NUREG/CR-6101 are related in this section to the major headings of the proposed vendor assessment procedure. The purpose is to demonstrate how the questions proposed by Lawrence [1993] can be used to achieve the objectives of Preckshot [1994]. Plan audits would be done at intervals as described above. An essential part of plan audits is keeping records of audits and tracking compliance and consistency with past performance.

In the following, NUREG/CR-6101 is referred to as "Lawrence," the section numbers refer to sections in that document, and the question numbers refer to questions in the cited sections. Tabular correlations between Lawrence questions and subjects and the vendor assessment steps are provided, along with accompanying narrative explanations. The steps "Evaluate Vendor History" and "Evaluate Current Program." are closely related and appear together in the tables. They address the same topics, but from retrospective and propsective viewpoints. The former step is concerned with past plans and their execution and the latter is concerned with proposed plans and the likelihood of their success as indicated by past plans and past results. The tables provide easily accessible numeric question number correlations, while the narratives provide more extensive explanations than are possible with tables. Corresponding tables and narratives are placed on opposing pages for convenient cross referencing.

4.1 Software Project Management Plan—SPMP (§4.1.1)

Table 2 shows a summary of the relationships between Lawrence and the vendor assessment steps with respect to topics covered by the SPMP. Further discussion is given below.

Determine Criticality

Lawrence suggests that the size, scope, and contents of the plan be appropriate to the level of safety criticality. Generally, criticality would be determined at project outset, and the amount of audit effort and scrutiny would be determined at that point.

Determine Organizational Interfaces

The questions noted in Table 2 examine organizational interfaces from various perspectives: organizational structure, boundaries, and defined interfaces; responsibilities for deliverables and project activities; and interactions with project support functions.

Evaluate Vendor History

This is an examination of: previous risk management activities, previous projects accepted, adequacy of planned reports and feasibility of planned staffing and training as indicated by the results of previously executed plans, and anticipated documentation be produced under the SPMP. The questions on methods, tools, and techniques should focus on records of approaches applied to previous products that answer the detailed technical questions in Lawrence and that provide an indication that the SPMP is feasible due to prior success.

Evaluate Current Program

Given the answers to "Evaluate Vendor History," are the planned milestones and deliverables feasible in light of demonstrated competence?

Check for Negative Factors

Do project priorities, as supported by pay and promotion history, match the stated commitment to safety? At the SPMP audit stage, only vendor history can provide answers to the other negative factor questions, except that answers regarding process model and staffing may indicate unrealistic schedules or resource allocations, or an underfunded effort.

Evaluate Commitments (or Execution of Commitments)

These questions regarding the SPMP represent commitments of resources and auditable deliverables. None of these are design commitments. No Lawrence question addresses commitments to adhere to promises made in the SAR or to Appendices A or B of 10 CFR 50.

Perform Periodic Reviews

At subsequent audits, the follow-through question measures how well the commitments in the SPMP are being followed.

Table 2. Mapping of SPMP Questions to Assessment Steps

NUREG/CR 6101	Vendor Assessment Step	NUREG/CR 6101	Vendor Assessment Step	NUREG/CR 6101	Vendor Assessment Step	Vendor Assessment Step	NUREG/CR 6101	Vendor Assessment Step	NUREG/CR 6101	Vendor Assessment Step	NUREG/CR 6101	Vendor Assessment Step
Section or Question Number	Determine Criticality Subject	Section or Question Number	Determine Organizational Interfaces Subject	Section or Question Number	Evaluate Vendor History Subject	Evaluate Current Program Subject	Section or Question Number	Check for Negative Factors Subject	Section or Question Number	Evaluate Commitments Subject	Section or Question Number	Perform Periodic Reviews Subject
4.1.1.1	Grading by Safety	1.g	Deliverables Responsibility	1.all	Previous Process Models	Process Model	1.all	Process Model	1.all	Process Model	13.a	Plan vs Actual
		2.all	Organizational Structure	6.all	Past Assumptions & Constraints	Assumptions & Constraints	1.h	Adequate Resources	8.all	Monitoring & Controlling		
		3.all	Organizational Boundary & Interface	7.all	Historical Risk Management	Planned Risk Management	5.all	Project Priorities	9.all	Staffing		
		4.all	Project Responsibility	8.all	Previous Monitoring & Controlling Activities	PlannedMonitoring & Control	9.all	Staffing	11.all	Software Documentation		
		12.a	Support Functions	9.all	Previous Staffing	Proposed Staffing						
				10.all	Methods, Tools, Techniques	Methods, Tools, Techniques						
				11.all	Historical Software Documentation	Planned Software Documentation						

4.2 Software Quality Assurance Plan—SQAP (§4.1.2)

Table 3 shows a summary of the relationships between Lawrence and the vendor assessment steps with respect to topics covered by the SQAP. Further discussion is given below.

Determine Criticality

Lawrence suggests criticality determination for determining size and scope of the SQAP. Answers to the general questions on size and scope are necessary to determine criticality, since they identify the software to which the SQAP applies.

Determine Organizational Interfaces

Most of the management questions determine the relation of the SQAP and responsible people to other activities. The definition of life cycle phases describes the temporal interface between SQA activities and other project activities.

Evaluate Vendor History

The questions on documentation and reviews and audits can be demonstrated by presenting examples of documents from a previous effort during the initial SQAP audit. Verification of answers regarding problem reporting and corrective action relate to prior history of problem detection, defect tracking, root cause determination, and resolution. During an SQAP audit, this appears to be the only way to check vendor SQA history.

Evaluate Current Program

Mandatory SQA tasks are examined while questions on requirements, design, and test describe life-cycle phase-specific tasks that Lawrence considers necessary for an SQA program.

Check for Negative Factors

No questions address requirements stability. Problem detection, defect tracking, root cause determination, and resolution are evaluated as well as independence and provisions for adverse SQA comment.

Evaluate Commitments (or Execution of Commitments)

The question on standards requires naming the standard to which the SQAP was written, if one was used. The remaining questions address audit commitments, task commitments for assurance objectives, and defect tracking commitments.

Perform Periodic Reviews

No specific requirement under Lawrence exists to revisit the SQAP, but presumably activities described need to be verified on subsequent audits.

Table 3. Mapping of SQAP Questions to Assessment Steps

NUREG/CR 6101	Vendor Assessment Step	NUREG/CR 6101	Vendor Assessment Step	NUREG/CR 6101	Vendor Assessment Step	Vendor Assessment Step	NUREG/CR 6101	Vendor Assessment Step	NUREG/CR 6101	Vendor Assessment Step	NUREG/CR 6101	Vendor Assessment Step
Section or Question Number	Determine Criticality Subject	Section or Question Number	Determine Organizational Interfaces Subject	Section or Question Number	Evaluate Vendor History Subject	Evaluate Current Program Subject	Section or Question Number	Check for Negative Factors Subject	Section or Question Number	Evaluate Commitments Subject	Section or Question Number	Perform Periodic Reviews Subject
1.all	Size & Scope	2.all (exc c,d)	Management	2.d	Historical Tasks	Mandatory Tasks	2.b	Independence	1.c	Standard	5,6,7,8,9	Areas for re-visit
		2.c	Life Cycle Phases	3.all	Historical Documents	Planned Documents	9.all	Problem Rptg / Corr. Action	4.all	Review & Audit		
				4.all	Historical Review & Audit	Planned Review & Audit			5,6,7,8	Task Commitments		
				5.all	Previous Requirements Review	Requirements Review			9.all	Problem Rptg / Corr. Action		
				8.all	Previous Test	Test						
				9.all	History of Problem Rptg / Corr. Action	Plans for Problem Rptg / Corr. Action						

4.3 Software Configuration Management Plan—SCMP (§4.1.3)

Table 4 shows a summary of the relationships between Lawrence and the vendor assessment steps with respect to topics covered by the SCMP. Further discussion is given below.

Determine Criticality

Lawrence suggests that safety-related software projects should have an SCMP. SCM is mentioned by Lawrence and Preckshot [1994] as a mandatory design factor.

Determine Organizational Interfaces

Organizational interfaces, responsibilities, and interfaces to suppliers and other projects are addressed directly.

Evaluate Vendor History

Existing SCM organizations provide a source of vendor SCM history. Historical policy, if such exists, and examples of status accounting, configuration audits, and reviews assure the reviewer of the organization's ability to carry out the plan and audit its implementation.

Evaluate Current Program

Plans for interface control, SCM policies, configuration identification, change control, change authority, and supplier control are all covered. This should be sufficient to evaluate the currently planned program.

Check for Negative Factors

Support for problem reporting and the change process as well as resource issues for SCM are addressed.

Evaluate Commitments (or Execution of Commitments)

SCM requires commitments in a number of areas. These include resource commitments, skill commitments, and items to be listed as committed configuration items. The latter has particular importance because it describes the scope of SCM application. Commitments to configuration identification, change management, and status accounting methods are necessary for implementation of the SCM process. Commitments to configuration audits are the only way an outside reviewer will have visibility into the SCM operation. Altogether, the Lawrence questions constitute a searching examination of commitments made in the SCMP. These questions should be revisited during periodic reviews to see how well the commitments have been carried out.

Perform Periodic Reviews

Periodic review is necessary to see if the SCMP is actually being followed.

Table 4. Mapping of SCMP Questions to Assessment Steps

NUREG/CR 6101	Vendor Assessment Step	NUREG/CR 6101	Vendor Assessment Step	NUREG/CR 6101	Vendor Assessment Step	Vendor Assessment Step	NUREG/CR 6101	Vendor Assessment Step	NUREG/CR 6101	Vendor Assessment Step	NUREG/CR 6101	Vendor Assessment Step
Section or Question	Determine Criticality	Section or Question	Determine Organizational Interfaces	Section or Question	Evaluate Vendor History	Evaluate Current Program	Section or Question	Check for Negative Factors	Section or Question	Evaluate Commitments	Section or Question	Perform Periodic Reviews
Number	Subject	Number	Subject	Number	Subject	Subject	Number	Subject	Number	Subject	Number	Subject
4.1.3.1	Required for safety	1.all	Organization	2.a	Existing CM organization	Existing CM organization	4.a	Resource Adequacy	1.b	Skills	11.a	SCM review
		2.all	SCM Responsibility	3.b-e	Historical Interface Control	Interface Control	5.c,d	Change & Rptg Procedures	4.a	Resources		
		3.a	Organizational Interfaces	5.all	Historical Policy	Planned Policy	7.e,l,m,n	Change Process	6 exc d,f,m	Config. Ident. methods		
		4.b	Coordination w other projects	6.all	Historical Config. Ident.	Configuration Identification			5.c, 7.a-h,l,n, 8.c	Change Mgmt commitments		
		10.all	Supplier Control	7.all	Historical Control	Change Control			8.all	Status Accounting		
				8.all	Historical Status Acctg	Status Accounting			9.all	Audits		
				9.all	Historical Audits	Configuration Audits						
				10.all	Historical Supplier Cntl	Supplier Control						

4.4 Software Verification and Validation Plan—SVVP (§4.1.4)

Table 5 shows a summary of the relationships between Lawrence and the vendor assessment steps with respect to topics covered by the SVVP. Further discussion is given below.

Determine Criticality

Lawrence §4.1.4.1 suggests that the size, scope, and complexity of the V&V effort should be governed by the safety-criticality level of software covered by the plan. This should have been determined under Lawrence §4.1.1. Lawrence also requires that the software covered by the plan be identified and checks the identification of the criticality of individual software items.

Determine Organizational Interfaces

Interfaces to other plans are directly addressed. Interfaces to other activities and delineation of the responsibilities of the various V&V participants are also important in this step.

Evaluate Vendor History

Lawrence requires SVVPs to be under CM, which may be a source of prior SVVPs for historical evaluation. Anomaly reporting questions can be examined to see if anomaly reporting from previous V&V efforts meets the reviewer's expectations. Management overview provisions and V&V results feedback to the development process can be queried for historical examples to prove adequacy.

Evaluate Current Program

Scope and application of planned V&V and resource commitments are addressed. Tools and techniques, schedule and resources, planning for the minimal tasks, and coordination of V&V activities with other development activities and the life cycle are also covered. In addition, precursor (before requirements analysis) activities and life cycle phase activities are addressed. Lawrence implies, but does not explicitly ask, a question that requires a direct tie between a defined life cycle, V&V activities, and other development activities.

Check for Negative Factors

Underfunded effort and schedule-driven negative factors are addressed. The defect tracking and anomaly reporting questions should provide appropriate coverage of the defect tracking negative factor.

Evaluate Commitments (or Execution of Commitments)

Software committed to V&V is covered as is description of resource and schedule commitments. Anomaly reporting and defect tracking commitments as well as specific vendor commitments to V&V tasks to be executed at life cycle phases and to minimal V&V tasks are covered. Materials to be evaluated are identified along with summaries, conclusions, and recommendations to be supplied. Management overview commitments, V&V feedback to the development team, and commitments to interface to the development and regulatory organizations as are investigated.

Perform Periodic Reviews

There is no specific Lawrence question for periodic update of compliance with the V&V plan. However, periodic reviews of V&V activities at life cycle milestones would address this. Also, no materials are specifically called out for audit save the installation configuration audit and a final report.

4.4 Software Verification and Validation Plan—SVVP (§4.1.4)

Table 5 shows a summary of the relationships between Lawrence and the vendor assessment steps with respect to topics covered by the SVVP. Further discussion is given below.

Determine Criticality

Lawrence §4.1.4.1 suggests that the size, scope, and complexity of the V&V effort should be governed by the safety-criticality level of software covered by the plan. This should have been determined under Lawrence §4.1.1. Lawrence also requires that the software covered by the plan be identified and checks the identification of the criticality of individual software items.

Determine Organizational Interfaces

Interfaces to other plans are directly addressed. Interfaces to other activities and delineation of the responsibilities of the various V&V participants are also important in this step.

Evaluate Vendor History

Lawrence requires SVVPs to be under CM, which may be a source of prior SVVPs for historical evaluation. Anomaly reporting questions can be examined to see if anomaly reporting from previous V&V efforts meets the reviewer's expectations. Management overview provisions and V&V results feedback to the development process can be queried for historical examples to prove adequacy.

Evaluate Current Program

Scope and application of planned V&V and resource commitments are addressed. Tools and techniques, schedule and resources, planning for the minimal tasks, and coordination of V&V activities with other development activities and the life cycle are also covered. In addition, precursor (before requirements analysis) activities and life cycle phase activities are addressed. Lawrence implies, but does not explicitly ask, a question that requires a direct tie between a defined life cycle, V&V activities, and other development activities.

Check for Negative Factors

Underfunded effort and schedule-driven negative factors are addressed. The defect tracking and anomaly reporting questions should provide appropriate coverage of the defect tracking negative factor.

Evaluate Commitments (or Execution of Commitments)

Software committed to V&V is covered as is description of resource and schedule commitments. Anomaly reporting and defect tracking commitments as well as specific vendor commitments to V&V tasks to be executed at life cycle phases and to minimal V&V tasks are covered. Materials to be evaluated are identified along with summaries, conclusions, and recommendations to be supplied. Management overview commitments, V&V feedback to the development team, and commitments to interface to the development and regulatory organizations as are investigated.

Perform Periodic Reviews

There is no specific Lawrence question for periodic update of compliance with the V&V plan. However, periodic reviews of V&V activities at life cycle milestones would address this. Also, no materials are specifically called out for audit save the installation configuration audit and a final report.

Table 5. Mapping of SVVP Questions to Assessment Steps

NUREG/CR 6101	Vendor Assessment Step	NUREG/CR 6101	Vendor Assessment Step	NUREG/CR 6101	Vendor Assessment Step	Vendor Assessment Step	NUREG/CR 6101	Vendor Assessment Step	NUREG/CR 6101	Vendor Assessment Step	NUREG/CR 6101	Vendor Assessment Step
Section or Question Number	Determine Criticality Subject	Section or Question Number	Determine Organizational Interfaces Subject	Section or Question Number	Evaluate Vendor History Subject	Evaluate Current Program Subject	Section or Question Number	Check for Negative Factors Subject	Section or Question Number	Evaluate Commitments Subject	Section or Question Number	Perform Periodic Reviews Subject
1.c	Software Covered	1.a,b	Interfaces to Other Plans	4.1.4.1	Old SVVPs under CM	SVVP under CM?	2.b,c; 3.e-g	Underfunding	1.c	Software Covered	9.b	Installation Audit
4.b	Criticality of SW Items	2a, 3.h-k	Organization & Responsibilities	1.c-f	Prior Scope	Scope & Application	3.d,i; a of 5-10	Defect Tracking	2.b,c; 3.e-g	Sched & Resource Commitments		
		4.a,b	Coordination w Other Tasks	2.b,c; 3.e-g	Historical Resource Commitments	Sched/Resource Commitments			3.d,i; part a. of 5-10	Anomaly & Defect Tracking Commitments		
				2.d-f	Previous Tools & Techniques	Tools & Techniques			5-10.all	Vendor Commitments		
				3.a	Historical min Tasks	Minimal Tasks			3.a,b	Vendor-minimal tasks		
				2.b; 3.a,j	Past V&V Coordination	Coordination of V&V			3.c	Evaluation & Summaries		
				3.d	Anomaly Rptg History	Anomaly Reporting Plans			3.h	Dev/Regulator Interface		
				3.j	Management Overview	Management Overview			3.j	Management Overview		
				3.k	Results Feedback	Results Feedback			3.k	V&V Feedback		
				4.b	Past Precursor Activities	Precursor Activities						
				5.all - 10.all	Past Life Cycle Activities	Life Cycle Activities						

4.5 Software Safety Plan—SSP (§4.1.5)

Table 6 shows a summary of the relationships between Lawrence and the vendor assessment steps with respect to topics covered by the SSP. Further discussion is given below.

Determine Criticality

Lawrence §4.1.5.1 suggests that the size, complexity, and scope of the SSP should be dependent upon the safety criticality level of the software product. The SSP is required to be under configuration management control.

Determine Organizational Interfaces

Organizational structure and interfaces to other software organizations are covered. The relationship to SCM and SQA activities as well as the interfaces to subcontractors are addressed.

Evaluate Vendor History

Previous examples of documentation can be examined for suitability. The safety program records activity should be investigated and an appropriate question to ask is "Can you present examples of previous safety program records?" Previous certification activities should also be investigated.

Evaluate Current Program

Authorities and independence, resources to be applied, training resources, and SCM are examined. SQA is required but safety responsibilities lie with the safety organization. The application of safety considerations to (software) tools is considered along with pre-existing software and subcontractor safety evaluation provisions. Planned safety certifications are assessed if that is part of the SSP.

Check for Negative Factors

The whistle-blower negative factor is addressed directly. Defect tracking and unstable requirements issues are addressed by concentrating on SCM change control mechanisms. Resources and training are described, which addresses the underfunded effort negative factor.

Evaluate Commitments (or Execution of Commitments)

All noted commitments should be investigated.

Perform Periodic Reviews

This item addresses periodic reviews directly, by requiring, or questioning, whether the SSP continues to be followed.

Table 6. Mapping of SSP Questions to Assessment Steps

Section or Question Number	Determine Criticality Subject	Section or Question Number	Determine Organizational Interfaces Subject	Section or Question Number	Evaluate Vendor History Subject	Evaluate Current Program Subject	Section or Question Number	Check for Negative Factors Subject	Section or Question Number	Evaluate Commitments Subject	Section or Question Number	Perform Periodic Reviews Subject
4.1.5.1	Grading by Criticality	1.all	Organization & Responsibility	1.d-f; 6.c, 8.b; 9.d,e;	Historical Independence	Authority & Independence	1.f; 4.b	Whistle-blower	1.all	Authority & Org Commitments	13.a	Review
		7.all	SCM	2.all	Historical Resources	Resources	7.a,b	Defect Tracking	2.all	Resource Commitments		
		8.all	SQA	3.all	Past Training Resources	Training Resources	2.all; 3.all	Underfunding	3.all	Training Commitments		
		11.all	Subcontract Management	5.all	Previous Safety Documentation	Planned Safety Documentation			5.all	Documentation Commitments		
				6.all	Safety Program Records	Planned Safety Records			6.all	Records Format & Retention		
				7.all	SCM History	SCM Plans			9.all	Tool Approval		
				8.all	SQA History	SQA Plans			10.all	Purchased SW Approval		
				9.all	History with Tools	Plans for Tools			11.all	Subcontractor Management		
				10.all; 11.all	Historical Handling of Subcontractors; Purchased SW	Subcontractors; Purchased SW			12.a	Certification		
				12.a	Previous Certifications	Certification Plans						

4.6 Software Development Plan—SDP (§4.1.6)

Table 7 shows a summary of the relationships between Lawrence and the vendor assessment steps with respect to topics covered by the SDP. Further discussion is given below.

Determine Criticality

Lawrence says simply that safety critical projects should have a software development plan. There is no suggestion that the SDP should be sized or scoped according to criticality. The SDP should be under configuration management control.

Determine Organizational Interfaces

There are no direct organizational interface questions. Products available at life cycle phase completions, as well as inputs are covered and milestones are related to the SPMP schedule.

Evaluate Vendor History

Previous software developments are examined to determine vendor practice.

Evaluate Current Program

Task timing and sequencing information of the current program should be evaluated against the SPMP and other activities documented by the SVVP, the SCMP, and the SSP. Information about adequacy of documentation is addressed. There are no questions that cover the following evaluation areas: commitment to quality, training, process measurement, risk management by the developers, resource allocation, or problem detection, analysis, and correction. Team coordination, which includes acquisition and use of V&V results, is not mentioned. Methods, tools, and rules are addressed.

Check for Negative Factors

Schedule is covered, but there are no resource allocation questions or training questions, so the underfunded effort and schedule-driven negative factors are indeterminable for the SDP. There are no risk management questions or discussion of reliability, so that the optimistic reliability claim and failure to meet predictions negative factors are indeterminable for the SDP. Likewise, there is no mention of use of V&V results, so that defect detection, analysis, and resolution, at least as far as the developers are concerned, is not determinable.

Evaluate Commitments (or Execution of Commitments)

Standards commitments, development schedule commitments, and technical documentation commitments are addressed. There are no resource commitment questions.

Perform Periodic Reviews

Follow-up audits of plan execution are examined.

Table 7. Mapping of SDP Questions to Assessment Steps

NUREG/CR 6101	Vendor Assessment Step	NUREG/CR 6101	Vendor Assessment Step	NUREG/CR 6101	Vendor Assessment Step	Vendor Assessment Step	NUREG/CR 6101	Vendor Assessment Step	NUREG/CR 6101	Vendor Assessment Step	NUREG/CR 6101	Vendor Assessment Step
Section or Question Number	Determine Criticality Subject	Section or Question Number	Determine Organizational Interfaces Subject	Section or Question Number	Evaluate Vendor History Subject	Evaluate Current Program Subject	Section or Question Number	Check for Negative Factors Subject	Section or Question Number	Evaluate Commitments Subject	Section or Question Number	Perform Periodic Reviews Subject
4.1.6.2	SDP Required	1.all	Inputs & Products of Life Cycle Phases	1.all; 4.all	Historical task timing	Feasibility of proposed task timing	1.all; 4.all	Schedule	3.all	Standards Commitments	6.a	Review
		4.all	Milestones & SPMP Schedule	2.all	Historical usage of methods and tools	Methods & Tools			4.all	Resource Commitments		
				4.all	Historical schedule performance	Feasibility of Schedule			5.all	Training Commitments		
				5.all	Historical Technical Documentation	Adequacy and Likelihood of Documentation						

4.7 Software Integration Plan—SIP (§4.1.7)

Table 8 shows a summary of the relationships between Lawrence and the vendor assessment steps with respect to topics covered by the SW Integration Plan. Further discussion is given below.

Determine Criticality

Lawrence suggests size, scope, and contents of the SIP should be determined by safety-criticality of application. No yardstick is given. The SIP is to be under configuration management control.

Determine Organizational Interfaces

The relation of the SIP to the personnel responsible for SCM is considered; the SIP and delivery of SIP products to the V&V effort are addressed. No other organizational interfaces are noted.

Evaluate Vendor History

Previous plans could be examined to determine the adequacy of previous efforts. Also, previous interactions with the SCM and V&V organizations could be investigated.

Evaluate Current Program

Technical details are revealed by some questions. Resource allocations (personnel) are covered, as is risk analysis and response.

Check for Negative Factors

Resource allocation and schedule are indeterminable. Contingency actions are covered; there are no questions about dealing with anomalies.

Evaluate Commitments (or Execution of Commitments)

There are no questions covering commitments to standards, schedule, or resources.

Perform Periodic Reviews

Determine if the SIP was followed.

Table 8. Mapping of SW Integration Plan Questions to Assessment Steps

NUREG/CR 6101	Vendor Assessment Step	NUREG/CR 6101	Vendor Assessment Step	NUREG/CR 6101	Vendor Assessment Step	Vendor Assessment Step	NUREG/CR 6101	Vendor Assessment Step	NUREG/CR 6101	Vendor Assessment Step	NUREG/CR 6101	Vendor Assessment Step
Section or Question Number	Determine Criticality Subject	Section or Question Number	Determine Organizational Interfaces Subject	Section or Question Number	Evaluate Vendor History Subject	Evaluate Current Program Subject	Section or Question Number	Check for Negative Factors Subject	Section or Question Number	Evaluate Commitments Subject	Section or Question Number	Perform Periodic Reviews Subject
4.1.7.1	Grading by Criticality	4.d,e	Link to SCM	1.all	Historical stepwise integration	Feasibility of planned integration steps	2.d,e; 4.c	Contingency Plans - Risk Management			5.a	Review Performance
		4.f	Link to V&V	2.a,b	Prior environment and tools	Adequacy of planned integration environment and tools	3.b,c,d	Stable Staffing				
				2.d,e; 4.c	Historical Risk Management and Contingency Plans	Adequacy of Risk and Contingency Planning						
				2.c	Historical Prioritization of Integration Activities	Appropriate priorities						
				3.b,c,d	Prior staffing	Adequacy of Personnel Allocations						
				4.all	Prior Integration Procedures	Adequacy and completeness of procedural planning						

4.8 Software Installation Plan—SIP (§4.1.8)

Table 9 shows a summary of the relationships between Lawrence and the vendor assessment activities with respect to topics covered by the SW Installation Plan. Further discussion is given below.

Determine Criticality

Lawrence suggests that the size, scope, and contents of the SIP should be determined by the safety-criticality level of the application. No yardstick is given. The SIP is to be under configuration management control.

Determine Organizational Interfaces

No organizational interface questions are given.

Evaluate Vendor History

Prior installation procedures and prior installation plans can be examined to determine how they worked.

Evaluate Current Program

Technical details are covered.

Check for Negative Factors

No installation defect or error tracking questions are given.

Evaluate Commitments (or Execution of Commitments)

There are no commitments to evaluate.

Perform Periodic Reviews

The success of one or more prior tests or attempts to implement the installation plan should be examined.

4.9 Software Maintenance Plan SMP (§4.1.9)

The proposed vendor assessment procedure did not cover the maintenance phase of the software life cycle. It was directed at vendors being assessed prior to and during production of a software product.

Table 9. Mapping of SW Installation Plan Questions to Assessment Steps

NUREG/CR 6101	Vendor Assessment Step	NUREG/CR 6101	Vendor Assessment Step	NUREG/CR 6101	Vendor Assessment Step	Vendor Assessment Step	NUREG/CR 6101	Vendor Assessment Step	NUREG/CR 6101	Vendor Assessment Step	NUREG/CR 6101	Vendor Assessment Step
Section or Question Number	Determine Criticality Subject	Section or Question Number	Determine Organizational Interfaces Subject	Section or Question Number	Evaluate Vendor History Subject	Evaluate Current Program Subject	Section or Question Number	Check for Negative Factors Subject	Section or Question Number	Evaluate Commitments Subject	Section or Question Number	Perform Periodic Reviews Subject
4.1.8.1	Grading by Criticality			1.all	Prior Installation Environments	Adequacy of Environment Planning					4.a	Review Dress Rehearsal of Installation
				2.all	Prior Installation Package Descriptions	Completeness of Proposed Installation Package						
				3.all	Prior Installation Procedures	Feasibility of Proposed Procedures						
				4.a	Prior Installation Plan Tests	Adequacy of Proposed Test of Installation Activities						

REFERENCES

ANSI/ASQC Q91-1987. "Quality Systems—Model for Quality Assurance in Design/Development, Production, Installation, and Servicing," American Society for Quality Control, identical to ISO 9000-1.

ASME NQA-1-1994. "Quality Assurance Requirements for Nuclear Facility Applications."

Bamford, Robert, and William J. Deibler, II. 1993. "A Detailed Comparison of the SEI Software Maturity Levels and Technology Stages to the Requirements for ISO 9001 Registration," SSQC, San Jose, CA.

British Ministry of Defence. 1991. *The Procurement of Safety Critical Software in Defence Equipment Part 1: Guidance*, Interim Defence Standard 00-55, 5 April 1991a.

British Ministry of Defence. 1991. *The Procurement of Safety Critical Software in Defence Equipment Part 2: Requirements*, Interim Defence Standard 00-56, 5 April 1991b.

Humphrey, Watts S. 1988. "Characterizing the Software Process: A Maturity Framework," IEEE Software, March, pp. 73–79.

IEEE 603-1991. "Criteria for Safety Systems for Nuclear Power Generating Stations."

IEEE-7-4.3.2-1993. "Standard Criteria for Digital Computers in Safety Systems of Nuclear Power Generating Stations."

International Organization for Standardization (ISO). 1993. *Baseline Practices Guide*, Spice Project, ISO/IEC JTC1/SC7/WG10, Issue 0.05, September.

ISO 9000-3. 1987. "Guidelines for the Application of ISO 9000-1 to the Development, Supply, and Maintenance of Software," International Organization for Standardization (ISO).

Lawrence, J. Dennis. 1992. Workshop on Developing Safe Software: Final Report, Lawrence Livermore National Laboratory, November 30.

Lawrence, J. Dennis. 1993. "Software Reliability and Safety in Nuclear Reactor Protection Systems," NUREG/CR-6101, UCRL-ID-117524, Lawrence Livermore National Laboratory (November).

Lawrence, J. Dennis. Letter to John Gallagher, NRC. Re: March 10 Meeting with TRW at LLNL. LLNL CS&R 93-03-20, dated March 24, 1993a.

Lawrence, J. Dennis. Letter to John Gallagher, NRC. Re: June 1 Meeting with CSC. LLNL CS&R 93-07-05, dated July 8, 1993b.

Lawrence, J. Dennis. Letter to John Gallagher, NRC. Re: July 23 Meeting with IBM. LLNL CS&R 93-08-01, dated August 2, 1993c.

Lawrence, J. Dennis and G. G. Preckshot. 1994. "Design Factors for Safety-Critical Software," NUREG/CR-6294, Lawrence Livermore National Laboratory (December).

Leveson, Nancy G. and Clark S. Turner. 1993. "An Investigation of the Therac-25 Accidents," *Computer*, July, pp. 18–41.

MoD, See British Ministry of Defence.

Paulk, Mark C., Bill Curtis, Mary Beth Chrissis, and Charles V. Weber. 1993. "The Capability Maturity Model for Software, Version 1.1," Software Engineering Institute, Carnegie Mellon University, CMU/SEI-93-TR-24, February.

Persons, Warren L. 1993. "Re: Visit with Hewlett-Packard Company, August 6, 1993," Letter to John Gallagher, NRC, LLNL CS&R 93-08-06, August 9.

Ploof, F. W. and G. G. Preckshot. 1993. "Critical Assessment of Software Design Factors," Lawrence Livermore National Laboratory (October). Attached as Appendix B.

Ploof, F. W. and G. G. Preckshot. 1993. "Subject: Assessment of Vendors (Task 16)," Letter to John Gallagher, NRC, LLNL CS&R 93-08-18, August 20.

Preckshot, G. G. 1993. "Appendix B: Real-Time Systems Complexity and Scalability" of NUREG/CR-6083, Reviewing Real-Time Performance of Nuclear Reactor Safety Systems.

Preckshot, G. G. 1993. "Assessing Commercial Off-The-Shelf (COTS) Software in Reactor Applications," Lawrence Livermore National Laboratory under contract to NRC, draft report under review.

Preckshot, G. G. 1994. "Commercial Off-The-Shelf (COTS) Commercial Dedication Process," Lawrence Livermore National Laboratory under contract to NRC, draft report under review.

Preckshot, G. G. 1994. "Vendor Assessment Process," Report, Lawrence Livermore National Laboratory. Attached as Appendix A.

Preckshot, G. G. and J. A. Scott. 1994. "Prospective Software Complexity Measures for NRC Use," Letter Report, Lawrence Livermore National Laboratory, May 27.

Scott, J. A., G. G. Preckshot, J. D. Lawrence, and G. L. Johnson. 1995. "Issues and Relationships Among Software Standards," Draft Report, Lawrence Livermore National Laboratory.

Appendix A

Vendor Assessment Process

Manuscript date: May 6, 1994

Prepared by
G. G. Preckshot
Lawrence Livermore National Laboratory
7000 East Avenue
Livermore, CA 94550

APPENDIX A CONTENTS

Vendor Assessment Process

A1.0 Introduction

This vendor assessment process is a proposed method for NRC reviewers to assess the capabilities and products from software vendors who develop safety-critical applications for nuclear power plants. This report, based on previous work [Ploof & Preckshot 1993, Preckshot 1993, Lawrence 1993], is intended to be accompanied by the presentation vu-graphs in the Appendix, and provides explanatory text thereto.

The reviewer faces potentially insurmountable obstacles, which can be overcome if the reviewer maintains realistic expectations of what can be accomplished. The following section defines five general cautions and introduces a seven-step process for vendor review.

A1.1 No Silver Bullet

There is no single method or tool that can either certify a vendor to produce reliable and safe software or verify that a particular software product is reliable or safe. The current state of the software art requires that multiple factors be examined and taken into account.

A1.2 Not a One-Stop Process

Assessment of software vendors and their products is not a one-stop process. Not only is software engineering a dynamic process, but the engineering and manufacture (e.g., programming) of a software product takes place over an extended time. The reviewer should expect to examine a candidate vendor at several points during software development.

A1.3 Quality is Perishable

Current ISO 9000 practice envisions re-certification at periodic intervals since organizations change over time, sometimes for the worse. A reviewer should expect an assessment to remain current only as long as personnel turnover, reorganizations, or financial pressures have insignificant effects.

A1.4 Insufficient Records

Insufficient records may be a hallmark of the software industry. The reviewer should expect that documentation might be missing or was never generated.

A1.5 Statistically Invalid Data

Performance claims based on anecdotal evidence are also a hallmark of the software industry. The reviewer should be especially vigilant regarding skewed or optimistic data reporting.

A2.0 Seven Steps to Assessment

This section introduces a seven-step assessment process that addresses the five cautions. It is based on previous work, as described above. The following sections present more detail on the reasons behind each step.

A2.1 Determine Criticality

Maintaining a high-quality software process and proving this to regulators can be extensive. This expense is unwarranted for software that has no safety impact; thus, the first step should be to determine whether costly review efforts by both the regulator and the vendor are appropriate for the products the vendor produces.

A2.2 Determine Interfaces

Interfaces between vendors, between subgroups within one company, or between hardware engineers or scientists on one hand and software engineers on the other have typically been a big source of errors and failures of software products. The important thing to remember is that interfaces are between people; at the root of any misunderstanding are two or more people. The reviewer should make sure that interfaces are identified, documented, and that conflicts are resolved.

A2.3 Evaluate History

Evaluate history to answer the question: "have you been doing what you say you will be doing?" A software vendor with proven history of good practice is likely to continue. Conversely, promises are easy to make but difficult to fulfill if they represent new practices. A reviewer evaluates a vendor's history to establish confidence in the vendor's claims.

A2.4 Evaluate Current Program

Evaluate the vendor's current software development program and plans for each product under review for (1) appropriateness to the role the product will play and (2) consistency with the vendor's past practices. A significant and rapid change in practices is cause for concern, and has been called "high-risk" by the participants in previous LLNL studies.

A2.5 Check for Negative Factors

Presentations and optimistic forecasts are easy to make. Check the vendor organization for indications that accompany significant software development failures in the past. Often, vendor management may not even be aware of these indications.

A2.6 Evaluate Vendor Commitments

The vendor should make appropriate commitments to standards and practices based on the safety role each software product plays. The reviewer must decide if vendor historic practices, current plans, and commitments make a convincing case that the vendor will produce (or has produced) safe and reliable software.

A2.7 Perform Periodic Reviews

Subsequent to the initial evaluation, the reviewer will need to perform periodic reviews to ensure that software quality does not perish, and that practices, plans, and standards to which the vendor has committed are being followed.

A3.0 Determine Criticality

A previous document [Preckshot 1993], describes a categorization scheme based on IEC 1226. Typically, two vectors need to be considered when determining software safety criticality: the safety role the ultimate software product plays, and the relation the product under review has to the ultimate product. The simplest example that illustrates the point is that of compilers. Compilers are not used directly in a safety system, but their output may be. The theory of criticality categorization proposed in Preckshot [1993] is that there are three relations the product under review can have to the ultimate product:

1 The product is used directly in a safety application.

2 The product directly produces a product used as in 1.

3 The product helps produce a product used as in 1.

"Helps produce" means that there are intervening or other methods that assure the safety and reliability of the ultimate product. The output of a software product as described in 2, above, has no intervening method of assuring its output. For example, compilers that produce machine-executable code to be used in safety applications.

In view of the relations proposed above, to complete the categorization scheme, it remains to determine what safety role the ultimate product plays. In a precis of Preckshot [1993], these roles are:

- Essential to safety
- Important to safety (display)
- Important to safety (can challenge the protective system)
- Important to safety (diverse actuation)
- Related to safety (surveillance or support)

A4.0 Determine Organizational Interfaces

Organizational interfaces between a software vendor and the supplier of a nuclear steam supply system (NSSS) can be complicated by the fact that the NSSS supplier and the software vendor are two different companies. However, in the large organizations extant in today's business world, this may be a superfluous distinction; departments in a single business have sometimes been as separated as if they were in different companies. Instead, it is probably more profitable to concentrate on areas where historically there have been problems. A selection of six interface areas is listed below:

- System/hardware to software requirements translation
- Software requirements configuration management and change control
- System safety requirements change control
- Product configuration management and change control
- In-plant delivery and subsystem validation
- Full plant certification testing

These interface areas can be explored by asking "who" questions, where "who" could be a title rather than an individual. Once the "who" is identified, the "what" and "how" can be determined. Representative "who" questions are:

- Who does the system engineering?
- Who understands nuclear steam supply systems?
- Who understands software and computer architecture?
- Who apportions system safety requirements to software and other systems?
- Who identifies software interfaces?
- Who maintains software requirements under change control?
- Who feeds back the implications of software design decisions to system engineering?
- Who is responsible for software configuration management and change control?
- Who is responsible for delivery?
- Who is responsible for plant integration testing?

In most cases, identifying the "who's" will lead to the organizational interfaces, or where the organizational interfaces should be, but aren't. In cases where an NSSS vendor is cooperating with a software vendor, the "who's" will involve persons in both organizations.

A5.0 Evaluate Vendor History

Evaluate vendor history to determine how past products have fared and whether the software vendor has experience developing highly reliable, safety-critical software. The reviewer should look for evidence of positive design factors and previous product successes that occurred because of good planning and execution. A significant finding of vendor history exploration can be that there is no history, or that documentation is so poor that the vendor cannot demonstrate past practices to any reasonable extent. Following the design factors of a previous report, a reviewer should look for:

- Commitment to quality as shown by past pay and promotion policy
- Consistent and thorough documentation policy
- Records of previous products showing - software requirements documentation - design documentation - Verification & Validation (V& V) - change control - interface documentation and control - delivered product configuration control
- Employee training program
- Use of life-cycle models
- Software development process measurement
- Process improvement
- Use of risk-management techniques
- Adequate resource allocation
- Early problem detection
- Defect tracking, root-cause determination, and resolution
- Current software maintenance activity
- Previous product deliveries
- Team coordination

A6.0 Evaluate Current Vendor Program

Evaluate the current software development program to assure that it is appropriate to the safety impact of the software the vendor proposes to develop or is developing. As previously mentioned, significant inconsistencies with past performance are cause for concern. There are six areas to evaluate (if the software project is in its early stages): organization, plans, software requirements, the use of a software life-cycle model, V & V, and opportunities for future audits.

A6.1 Organization

The vendor's organization structure should support the design factors discovered during historical evaluation. The organizational aspects most often mentioned by standards, experts, and surveyed software companies are independent of quality assurance functions (V & V, configuration management, problem tracking) from software production functions. After independence of the main units, the reviewer should look for organizational units responsible for employee improvement, process measurement and improvement, documentation, product delivery, and software maintenance.

A6.2 Plans

There should be plans for developing the current product that describe who will do what, and when. Lawrence [1993] §3.1 contains detailed suggestions for planning. A typical roster of plans covers the following areas:

- Software development
- V&V
- Software quality assurance
- Training
- Configuration management
- Documentation
- Safety

Other plans are made as software design and development progresses:

- Unit test plans
- Integration test plans
- Installation and checkout plans
- Operation and maintenance plans

A6.3 Product Requirements

Over 50% of software errors are requirements errors. At a minimum, vague, incorrect, or incomplete requirements have been implicated publicly in failures of software in operation and failure of a development team to complete a satisfactory product.

During early or initial reviews, a regulator will have a significant positive impact on product quality by demanding to see clear, complete, and unambiguous requirements for the product the vendor proposes to develop. During subsequent reviews, stability of these requirements—and traceability of product attributes to these requirements—are positive indications.

A6.4 Life Cycle

A product life-cycle model has become common in most engineering disciplines. Typical stages include conception, design, implementation, integration and test, delivery, operation and maintenance, and retirement. Most experts endorse a life-cycle model and applicable software standards because they provide a framework for coordinating the many activities required to produce a modern software product as well as milestones for measuring progress. The latter are directly important for the reviewer; life cycle milestones signify both the completion of activities that must be audited and the availability of auditable work products. Life cycle milestones should also be reasonably coordinated with other participants in the NPP construction or retrofit.

A6.5 Verification & Validation

V & V work products will provide the reviewer with the primary assurance that a particular product meets its requirements. The vendor's software quality assurance program is responsible for since the V & V plan is carried out, and that work products are produced. Anomalies discovered during V & V activities are traced, root cause is determined, and resolved. The records of this activity give the reviewer a direct visibility into the quality of the developing product. Consequently, the reviewer has significant interest in V & V plans and the vendor's current V & V program.

A6.6 Opportunity for Audits

An audit point has three important criteria:

- A reasonable, but not enormous, amount of work has been accomplished
- Viewable and consequential work products are available
- A significant milestone has been reached

For software products, these criteria are normally satisfied at the following times:

1. After requirements are done, but before design
2. At the completion of top-level design
3. At the completion of detailed design
4. At the completion of implementation
5. After system integration and validation
6. After each delivery and plant installation

During preliminary assessment of the vendor, the reviewer should select audit points consistent with the vendor's life cycle, plans, and V & V activities. The above candidate times are suggested as starting points.

A7.0 Check for Negative Factors

Even though a vendor may apparently have all appropriate organizational, historical, and planning factors in place, a company is an organization of people, and there may be symptoms that historically have accompanied software project failures. A selection of potential negative factors is listed below. Any vendor may at times encounter one or more of these factors. Consideration should be given to the preponderance of such factors, rather than concentrating on a single one.

- High personnel turnover
- Schedule-driven rather than quality-driven
- Short history
- Unstable requirements
- Grossly optimistic reliability claims
- Failure to meet predictions
- Failure to track defects and causes
- Underfunded efforts
- Hostile treatment of "whistle blowers"

A8.0 Evaluate Commitments

The vendor may make commitments to adhere to plans, to make work products available for review, follow certain practices or standards, or meet interface requirements with other suppliers, clients, or regulators. Commitments may be in the areas of:

- Quality assurance activities
- Use of certain standards or criteria
- Future configuration control
- Documentation
- Organizational interfaces
- Independence of certain quality assurance activities
- Process measurement and improvement

The reviewer should evaluate these commitments with regard to effect on future audits, legal design criteria (i.e., 10CFR50 Appendix A), or endorsements in relevant publications (e.g., Reg Guides, NUREGs, or Branch Technical Positions). No list of references is provided, since some of these publications are presently undergoing review.

A9.0 Perform Periodic Reviews

The vendor should be audited at planned audit points to ensure that:

- No additional negative factors have arisen
- The vendor's actual performance approximates planned performance
- The vendor adheres to commitments
- Essential design factors (as determined by historical evaluation) continue to be true
- Planned work products meet requirements
- Product design is appropriate to product role (product design factors)

Appendix B

Critical Assessment of Software Design Factors

Manuscript date: October 15, 1993

Prepared by
F. W. Ploof and G. G. Preckshot
Lawrence Livermore National Laboratory
7000 East Avenue
Livermore, CA 94550

Prepared for
U.S. Nuclear Regulatory Commission

Abstract

This report is a critical assessment of collected design factors lists from FIN L-1867 Task 2 and Software Engineering Institute (SEI) and the International Organization for Standardization (ISO) documents. "Design factors" are those indicators associated with an organization or a product which are thought to predict product reliability in a safety-critical application. The Task 2 design factors are individually assessed and then compared in groups with the SEI/ISO factors. The factor groups are then reviewed for applicability by the NRC.

APPENDIX B CONTENTS

Executive Summary

Design factors taken from FIN L-1867 Task 2 and from analyses of Software Engineering Institute (SEI) and the International Organization for Standardization (ISO) documents are assessed and compared in groups. The factor groups are then reviewed for applicability by the NRC based on the reputed effect of each factor group, observability, and pertinence to NRC practices and procedures. The report concludes that there are 15 significant process factor groups under these criteria, with two important auxiliary groups (negative factors and product factors). Eight other factor groups were judged to be of lesser effect, unobservable, or inappropriate to NRC practices. An important theme running through almost all design factors is the length of time required for capability improvement, and for certification that the principles described by the design factors are consistently and correctly applied. The sources are unanimous: a quality software development organization cannot be assembled overnight.

Critical Assessment of Software Design Factors

B1.0 Introduction

This report satisfies Part I, subpart (2) of FIN L-1867, Task 16, critical assessment of collected design factors lists. "Design factors" are those indicators associated with an organization or a product which is thought to predict product reliability in a safety-critical application. Several sets of such factors have been collected in the course of this task and Task 2 of the referenced FIN, and this report reconciles and assesses these factor lists in light of the NRC's regulatory mission.

B1.1 Contents of the Paper

This paper consists of four main sections and two appendices. The introduction describes the data sources from which design factors were accumulated and makes some comments on source biases. It also describes two categorization schemes (the design factors themselves are listed in Annex A). The second section of the paper discusses considerations that may have influenced sources in their choices of factors. The third section compares the SEI/ISO design factors with the Task 2 design factors. The final section ranks and assesses factors for use by the NRC in view of its role and customary practices. Annex A contains a listing and assessment of the Task 2 design factors. The SEI/ISO design factors previously reported (Ploof & Preckshot 1993) are re-listed for the convenience of readers. Annex B consists of the Task 2 and the SEI/ISO design factors compared in tabular form.

B1.2 Sources

As well as from subpart (1) of the current task (Ploof & Preckshot 1993), "design factors" have been identified and collected from various sources as mentioned below.

B1.2.1 SEI

SEI, or the Software Engineering Institute, is a DOD-funded research organization at Carnegie Mellon University that does research into characteristics that mark organizations that produce quality software. The DOD's interest is in improving the quality of software produced by its contractors. SEI has published a Capability Maturity Model (CMM) that postulates five levels of "maturity" in software-producing organizations. Methods of determining these maturity levels have been proposed and implemented and applied to a number of subject organizations. SEI currently does not address specific application domains, specific software techniques, or personnel-related concerns (Paulk et al 1993).

B1.2.2 ISO (9000-Series Standards)

ISO (International Organization for Standardization) is a consensus standards organization of which the United States, through the American National Standards Institute (ANSI), is a participatory member. ISO's influence is primarily European and South American, but influence in the U.S. is increasing. The ISO 9000 standards represent a consensus effort to provide quality standards by which software customers, among whom are European governmental entities, may judge potential software suppliers. ISO 9000-1 is a general quality assurance standard that is applicable to both hardware and

software production. ISO 9000-3 is a guide to applying the general requirements of ISO 9000-1 to software products.

B1.2.3 IEEE

The Institute of Electrical and Electronic Engineers is an international professional organization with significant influence in the United States. The IEEE has been very active in software as well as electrical and electronic consensus standards, and is the largest presence in the U.S. in this area. The IEEE often issues cooperative standards with ANSI.

B1.2.4 LLNL Task 2

Under the statement of work for Task 2 of the referenced FIN, Lawrence Livermore National Laboratory (LLNL) conducted two kinds of data collection activities for finding design factors.

Experts Conference

A conference of four internationally known software safety experts was convened and conducted by LLNL in San Diego, July 22–23, 1992. The conference report (Lawrence 1992) serves as a source of design factors.

Visits to Developers

Visits to four companies involved in commercial software development of appropriate risk and complexity were conducted by LLNL (Lawrence and Persons) during 1993. Trip reports and private communications with Dr. Lawrence were used to develop additional design factors (Lawrence 1993a, 1993b, 1993c, Persons 1993).

B1.3 Data Source Biases

Each of the data sources has biases that skew design factors towards the source's economic, academic, or philosophical interests. These are identified, as far as possible, below.

B1.3.1 SEI

SEI is primarily a research organization tied to an academic institution, and is conducting research based on a theoretical model of the software development process and organizations that develop software. SEI states that the work is oriented toward large organizations in aerospace or defense. The major bias in the SEI work is that the Capability Maturity Model (CMM) has been postulated, but not proven, to describe the software development process and the evolution that it undergoes as it is "improved" (Humphrey 1988). The philosophy behind the CMM is that, like ordinary manufacturing processes, the software development process can be measured and brought under statistical control. The CMM views product attributes, application areas, and personnel matters indirectly as outcomes of the software process. This results in an approach considerably different from traditional engineering design review, in that the organization and the production process is reviewed, not the product design. The theory is that a mature software development process will produce better software products. A significant feature of the CMM is the state memory (the maturity level) within the model; the same practice followed by two organizations at different maturity levels does not have the same import.

B1.3.2 ISO

ISO is an international consensus standards organization that produced the 9000-series standards for quality assurance. 9000-series standards must be commercially practicable (a reasonable percentage of software suppliers must be certifiable to gain consensus), so these standards are forged by agreement

between a wide range of quasi-governmental entities, software suppliers, and potential customers. The field of application for the 9000-3 standard is general software development, usually controlled by contractual agreements between a customer and a software developer or supplier. The theory of the ISO standards is similar to the theory behind the CMM; certificate of conformance with the ISO standards is supposed to increase the customer's confidence that a supplier will be able to fulfill contractual terms because good practices and an orderly software development process are being followed. There are significant differences between SEI and ISO, among which is that ISO has no equivalent to the CMM and maturity levels.

B1.3.3 LLNL Task 2

LLNL Task 2 was biased from the outset with the presumption that a set of "design factors" existed and could be determined. No other source makes this formal identification or uses this model, and the state-free concept of design factors may be in direct contradiction to the state-memory concepts implied by the CMM maturity levels. In any event, design factors as developed during LLNL Task 2 bear little resemblance to traditional engineering design factors, which are almost always attributes of the design itself, not the organization that performed the design.

The Experts

The four experts invited by LLNL represent four personal, although educated, views of software reliability and safety in critical applications. Each participant was biased toward his or her study area and experience, which can be determined by surveying his or her output of technical papers. None of the experts specifically endorsed or used a "design factors" model, but use of mathematical probability models and hazards analysis was a prevalent viewpoint. Each of the experts was a well-established senior faculty member or senior person with a government agency. Risk was viewed as the perceived probability of software failing and causing an unsafe condition or accident. One person in particular, Littlewood, uses Bayesian *a posteriori* statistics to compute the effect of testing failures on *a priori* perceptions of failure probability.

The Software Development Companies

Each of the companies visited has been successful in producing software under budget and schedule constraints for applications of significant technical difficulty and safety impact. Three out of the four developer organizations surveyed were large-scale aerospace or government contractors. Management in each company is aware of the SEI CMM and endorses it to some extent, in one case wholeheartedly. This resulted in views that focused on solution of each company's particular software development problems in each company's business area with an SEI CMM flavor. Orientation was typically that of a large commercial organization supplying software services or software-containing products to a technically- or government-oriented market.

B1.3.4 IEEE

The IEEE is a consensus standards organization that has a history of endorsing, with minor changes, *defacto* electronic standards championed by one or more financially interested businesses. Standards committee members are unpaid volunteers and are usually limited by personal economic considerations to company-supported representatives or academics. In our opinion, software standards to date have been more influenced by academic participants than by commercial participants, perhaps because no immediate economic consequences were foreseeable.

B1.4 Categories for "Design Factors"

As part of the process of assessing design factors, organizing them into categories is an aid to reasoning about them. From the above sources, we accumulated a consolidated list of design factors under the following headings (including added factors not addressed by the first four sources).

- General Factors
- Process Control Factors
- Management Factors
- Personnel Factors
- Development Factors
- Reliability and Safety Factors
- Negative Factors
- Product Factors
- Other Factors.

A full list of factors, including a one-paragraph explanation of each factor, is in Part 1 of Annex A.

The statement of work requires that factors be listed under the following two headings:

- Management Factors, and
- Technical Factors.

Factors are listed under these headings in Part 2 of Annex A.

B2.0 Influences on Factor Choices

In general, which design factors are considered important is influenced by point of view, organization, and business. The "known biases" in sources have already been mentioned, but without an intellectual framework relating the purported biases to facts observable about the sources. This section provides such a framework.

B2.1 Point of View

The point of view of a commentator on design factors depends on objectives, knowledge, and position. We identify five pertinent points of view in the data studied: the developer, the standards organization, the customer, the academic, and the regulator.

B2.1.1 The Software Development Company

The developer's objectives, as an organization, are to produce software for customers at the lowest financial risk and greatest profit for resources expended, although not at the expense of losing future business. The developer therefore has a utilitarian view of design factors. "Risk" for a developer means failure to deliver or delivering an unprofitable product, which differs significantly from the viewpoint of a regulator. Within a development organization we identify, again not exhaustively, three idealized job functions that have somewhat different points of view: programmer/analysts, certifiers, and managers.

The Programmer/Analyst

Programmer/analysts produce requirements, analyses, designs, and code, and are typically rewarded for meeting scheduled production. They are intellectually rewarded by solving difficult problems. Risk, for a programmer, is not delivering certified designs or code as promised.

The Certifier

Certifiers plan and perform analyses, tests, and certifications. Certifications that are produced on time are generally viewed with approval, but failure to certify designs or code is antithetical to the interests of both programmers and management, and is the main reason for requiring independence of testing, validation and verification (V&V), and quality assurance activity from production activity. A certifier's view of risk can vary, depending upon external pressures. In organizations in which certifiers are not insulated from producers, the risk may be in failing to deliver certification. In more independent circumstances, risk is viewed as certifying something that has hidden flaws.

The Manager

Management produces plans to allocate resources sufficient to meet scheduled business objectives, and monitors and controls work to ensure that objectives are met fully and on time. Management is rewarded for achieving objectives as scheduled or sooner at the least possible cost. A manager views risk in terms of not meeting planned objectives on time, or exceeding budget.

B2.1.2 The Standards Organization

Most standards organizations produce standards by consensus between participants selected from government, industry, and academia. These standards compete in an arena that requires general applicability and visible evidence that a standard is being met. There may be additional bias due to the industry group the standards organization serves; for instance, SEI is primarily targeted at large defense contractors. The view of the standards organization is one of conflict resolution, with the objective of getting agreement on a workable technical standard among interested parties. In general, any successful standard is a compromise. The standards organization views risk as failing to achieve consensus.

B2.1.3 The Customer

The customer for software products is generally, but not always, less sophisticated than the software developer in matters concerning software. The customer's objectives are to obtain a software product that fulfills the customer's needs at reasonable cost and prompt delivery. Customers view risk as schedule delays, cost overruns, and inadequate or "buggy" products. An often unperceived risk (by both customers and developers) is that customer needs do not always translate accurately to software requirements.

B2.1.4 The Academic

Academic objectives are varied and depend upon the research field and interests of a particular person. Depending upon seniority and tenure, academics can be influenced by career risk, effects on reputation, or ability to attract and retain consulting contracts. Persons in more secure positions tend to be more independent, but even untenured junior faculty members are usually ethical and attempt to take unbiased views dependent strictly on technical merit. A common complaint among the non-academic engineering profession does not regard ethics at all: academics are sometimes viewed as having little practical experience and as being too theoretical.

B2.1.5 The Regulator

The objectives of regulators such as the NRC are to prevent practices that may lead to unacceptable consequences, such as public exposure to radiation. Software is only one of many components whose failure may lead to such consequences, and the regulator is concerned in the end that the system shall not fail, even if individual components do. The regulator views risk, therefore, in a system context, and this viewpoint is more compatible with the academic viewpoint than the developer's viewpoint.

Regulators also balance risk assessments against national priorities and legal mandates, rather than commercial return.

B2.2 Organization

Organization affects design factor perceptions by organizational size, purpose, and customs.

B2.2.1 Size

Size of an organization affects resources available, speed of response, and internal communications. Generally, larger organizations have more resources, slower speed of response, and more levels through which internal communication must pass.

Suborganizations

A particular effect of size is that there may be several software-producing suborganizations within a large organization, and each may differ in local purpose and customs, often significantly enough to affect software quality. The position of a suborganization within a parent organization often has significant effects on resource allocation and status.

B2.2.2 Purpose

Organizational or suborganizational purpose may govern the importance of software development methods and management. For instance, an aerospace company may raise software management to high levels of importance because software safety is of prime importance to legal liability and company products. A utility may have a programming department that competes for resources with other maintenance departments because upper management perceives only a weak connection between power production and software maintenance.

B2.2.3 Customs

Organizational customs govern both the general environment in which a software suborganization must function, and particular software development methods that are favored. In consumer software development companies, it is usually customary to schedule product releases based on market conditions and acts of competitors. This leads to schedule-driven development with requirements set by market research, not always a stable oracle. One noted consumer software developer has a corporate culture based on the irregular and workaholic work habits of its Chief Executive Officer (CEO), and experiences high staff turnover as a result. The space shuttle flight software prime contractor has a more deliberate and structured corporate culture and experiences low staff turnover.

B2.3 Business

The business a company or organization is in can affect the relative importance of certain design factors because of wide variation in the level of software reliability required to compete in the business sector.

B2.3.1 Commercial Software

Products produced for the commercial (e.g., business or consumer) market generally require only enough reliability so that customer gripes remain at a manageable level. It is customary practice in this business sector to release buggy products because the earliest product out will usually acquire the largest market share. High-reliability design techniques are not important to this sector.

B2.3.2 Commercial Development for Hire

Commercial development is done by software consultants working under contract and has a varied history. The genesis of both the SEI maturity rating system and the ISO 9000 standards has been the unpredictable quality coming from such consultants or software development companies. Importance of high-reliability design factors may depend upon previous contracts.

B2.3.3 Peripherally Involved

A peripherally involved business is one that uses software, but does little development of it. This would include some electric utilities, some service organizations, some manufacturers, and similar businesses. These businesses are customers for software products and may be unaware of the importance of any software design factors.

B2.3.4 Industrial Process Control

Suppliers of industrial process control systems and software-containing components such as Programmable Logic Controllers (PLCs) must supply at least moderately reliable equipment for real-time control. Major players in this business regard reliability design factors as important.

B2.3.5 High-Risk Technical

Most organizations in aerospace or other high-risk technical businesses are well-acquainted with high-reliability software development methods and design factors. However, some high-risk businesses, such as medical equipment manufacturers, have had public failures that demonstrated ignorance of even rudimentary good practice (Leveson & Turner 1993).

B3.0 Comparison of Design Factors

The design factors listed in parts 1 and 3 of Annex A were grouped in comparison groups in the table of Annex B. In the following, the factors in each group are assessed as to why they were proposed and what influences may have prompted the various sources to propose them or ignore them. Then, factors from other sources not present on either list are discussed.

B3.1 SEI/ISO versus Task 2

In general, factors gleaned from the two standards organizations[1] were more general, and favored criteria that provided visible evidence that required attributes were present. Developer organizations were more concerned with day-to-day utilitarian factors that affected product output. The Task 2 experts took good software methods as a given, and were more directed toward risk estimation and hazards analyses.

B3.1.1 Quality Commitment Group

A commitment to quality by management, or alternatively everybody, is regarded as necessary to produce good software. Developer organizations had more specific, practical applications of the quality commitment principle, of which the most significant is probably that the organizational reward structure must match the quality commitment.

[1] SEI is viewed here as a *defacto* standards organization.

B3.1.2 Policy and Documentation Group

The standards organizations concentrated on documentation as a means of demonstrating that policies, plans, organizational structure, and product are defined and under management control. The developer view was more utilitarian and was directed toward (apparently) deliverable product documentation and documentation necessary for internal control. The developers specifically mentioned software interfaces as items requiring documentation and control, and their view is repeated by both ISO and SEI documents.

B3.1.3 Personnel Qualifications Group

Personnel qualification was not directly addressed by SEI, and in fact was excluded by Paulk (Paulk et al 1993). ISO 9000 has several requirements for "qualified" personnel and for personnel training, which can reasonably be interpreted to mean personnel of high intellectual capability. The developer's viewpoint reflected the prevalent literature view that the single most important factor in producing quality software is the intellectual capability of the persons producing it. From the emphasis by developers, finding high-quality software development personnel continues to be a priority.

B3.1.4 Independence Group

The deleterious effect of having certifiers under control of the same management responsible for product development appears well known to all parties. There are no significant differences on this point.

B3.1.5 Independent Review Group

Independent review is regarded as useful, although probably for different reasons. ISO, in particular, makes certification by independent agencies a part of the ISO 9000 process. Developers were more concerned with calibration of management perceptions against outside viewpoints.

B3.1.6 Resource Group

The standards organizations require that appropriate resources and training should be provided to personnel for certification or higher maturity levels. None of the developers or the Task 2 experts mentioned this point, possibly because they thought it was obvious. This speculation would be less appropriate with organizations of lesser resources and reputations.

B3.1.7 Team Coordination Group

The issue of large team or intergroup coordination was addressed directly by SEI and ISO, and idiosyncratically by developers. The developers are aware of the problems of managing large teams, but their specific comments addressed only narrow issues because of the focus of the developer interviews.

B3.1.8 Process Improvement Group

Process improvement is a major part of the SEI Capability Maturity Model and so is addressed extensively by that organization. ISO has only recently begun action on process assessment with the Spice Project (ISO 1993), and the maturity states inherent in the SEI model still represent a significant difference between SEI and ISO. However, ISO favors process improvement and requires periodic assessment for process improvement purposes. Among the Task 2 factors, a majority of detailed individual process improvement factors were due to one developer that is rated at SEI maturity level 5, although other developers contributed two factors. Other than the state assumptions of the CMM, there are no significant disagreements on this factor group.

B3.1.9 Process Measurement Group

As part of the process improvement process, a measurement program is required by SEI. The SEI maturity-level-5 developer contributed design factors from its practices of process measurement and stabilization. This level of detail and specificity is unique to this organization, although the advice given in the statement of the design factors appears reasonable. ISO has a considerably less-developed approach to process measurement, and, although such activities are not ruled out by ISO 9000 standards, they are also not required in detail. A data base for process metrics is required, but with few exceptions, ISO 9000-3 does not spell out what measurements must be made.

B3.1.10 Development Risk Group

ISO and SEI require that management estimate work effort and correct the effects of mis-estimation, which is essentially the management of development risk. The Task 2 experts did not address development risk. The software developers were directly concerned with development risk, although their primary concern may have been more for commercial viability than conformance with standards. Regardless, it will still be appropriate for regulators to review development risk performance of software developers under their regulatory purview because poorly managed risk means that "non-essentials," such as quality, are sacrificed to expediency.

B3.1.11 Life Cycle Group

Although there were differences in detail, there is no disagreement on the utility or the necessity of using an appropriate life cycle model as a framework for development and quality management activities.

B3.1.12 Requirements Group

There is no disagreement regarding the importance of clear, stable, and validated requirements. All parties regard such requirements as crucial to software quality.

B3.1.13 Development Attribute Group

The standards organizations describe the necessary attributes of software designs and development. The developers evidently did not comment because they believed that their designs and methods possessed these attributes. This speculation would be less appropriate with organizations of lesser resources and reputations.

B3.1.14 V&V Group

The standards organizations require that software be independently tested and validated. Developers offered detailed advice regarding types of testing, V&V planning, and product design for V&V. Independence is covered in the independence group. There is no significant disagreement.

B3.1.15 Product Delivery Group

ISO, being particularly oriented toward commercial or consultant software development, had requirements for reliable delivery of product. The product delivery process was addressed at length by the SEI level 5 developer, which has an extensive and lengthy procedure for ensuring accurate and controlled product delivery. One other developer, a contract consulting firm, also has explicit delivery procedures. The other two developers did not comment.

B3.1.16 Software Maintenance Group

As with product delivery, ISO considered the maintenance phase after software delivery as a responsibility of ISO-9000 certified developers. No other source considered software maintenance explicitly.

B3.1.17 Configuration Management Group

There was no disagreement on the need for or importance of configuration management. It is regarded as crucial. One configuration management subject was consistently and particularly required: change control.

B3.1.18 Tools Group

The use of tools is considered a positive factor, but few specific recommendations were given by any source. Reviews, walkthroughs, and inspections were recommended. Prototyping and simulation were recommended.

B3.1.19 COTS Group

Certification of commercial off-the shelf software (COTS) products used in development is considered a positive factor by the standards organization. The details of COTS certification, however, are still subjects of controversy. An inadequately done certification may be a negative factor because it results in unwarranted confidence in a COTS product.

B3.1.20 Contracts Group

The ISO 9000 certification process is constructed for software developers offering services for hire to interested clients. The standards contain certain provisions with respect to contractual performance that have little to do with product quality, except for contractual agreements to clarify software requirements. This is redundant considering the requirements group discussed above. SEI also has provisions for subcontractor management that would be typical of a large defense prime contractor. These provisions require that the prime contractor ensure that the subcontractor meets the same quality assurance criteria that the prime contractor is required to meet.

B3.1.21 Miscellaneous Opinions Group

Miscellaneous opinions relate to a particular source's business situation and are judged to be too narrow to be used as general design factors. Whether the particular opinions expressed in this group are significant design factors is open to debate. One of the opinions describes a characteristic (adaptability) of an SEI maturity level 5 organization, but in isolation. This characteristic would appear to have little meaning outside the CMM.

B3.1.22 Early Problem Detection Group

Early problem detection is the early identification or detection of critical components, errors, or complex designs so that these problems can be addressed while the cost of solving them is still small. This is a specific development technique favored by cost-conscious managers that is not directly addressed by SEI or ISO except under risk management. Consequently, neither SEI or ISO include this specific item in their requirements, while some developers regard this as a very significant process factor.

B3.1.23 Defect Tracking Group

There is no significant difference on defect tracking, but the reasons for it may vary between standards organizations and developers. The standards organizations favor product review and corrective action plans as a systematic way of recording and improving product quality and the development process, while the developers additionally intend to reduce business costs by not making the same mistake twice.

B3.1.24 Reliability Practices Group

Neither the standards organizations or the developers contributed much to this group; these individual factors were provided almost exclusively by the four experts at the San Diego meeting. The reason for this is probably that the reliability advice is either too specific or too controversial at this time to be addressed by general standards or to achieve consensus. Nonetheless, for the relatively small field of ultra-high reliability, these factors are significant.

B3.1.25 Negative Factors Group

Negative factors were proposed exclusively by the developers surveyed and probably represent specific bad experiences. It is not customary for standards to list bad practices, but usually good or preferred practices, so that it is no surprise that there was no standards organization participation in this group. For a regulator reviewing a potential software participant in a nuclear reactor, negative factors will be useful.

B3.2 Other Factors not in Either List

With the exception of the reliability practices group, the design factors extracted from SEI and ISO publications and those taken from Task 2 are primarily "process" factors. That is, they apply to organizational or management practices with the underlying assumption that a software organization meeting the factor criteria will produce quality software. Strictly speaking, these are not "design factors," since they are not directly associated with the design of a particular product. A small set of design factors that can be associated with a particular product were identified.

B3.2.1 Product Factors Group

The design factors in this group were taken from MoD (1991) and Preckshot (1993). When present, they indicate that extremely simple design practices are being used. This is generally regarded as positive for ultra-reliable software.

B4.0 Assessment of Factor Utility for Regulation

The design factor groups as assessed above were reviewed for their utility to the NRC and ranked in two groups, with comments, below. Refer to Annex A for details on Task 2 factors or the factor lists. Rankings within the essential group should be viewed as approximate, since all the essential design factors should be present. The design factors are stated for ultra-reliable software, where it is assumed that the highest quality software is to be developed. Product factors are listed separately and should be applied on a per-product basis. Negative factors are listed separately and should be used as a consistency check by NRC reviewers.

The criteria used to rank factors were the reputed effect of each factor, observability, and pertinence to NRC practices and procedures. Under the heading of reputed effect, factors were reviewed for effect on "quality" and connection to safety requirements. Quality, unfortunately, is a slippery term that sometimes means "meeting contractual requirements," "meeting software requirements," or rarely, "solving the system safety problem." Factors that are observable either through documentation

inspection, limited company review, or product inspection were favored. Factors that could be discovered by the NRC practices of periodic, scheduled, or milestone inspections, either by NRC personnel or contracted, independent reviewers, were also favored.

B4.1 Essential Design Factors

1. Quality Commitment

A commitment to quality is the most important factor. The organization's reward structure should match the quality commitment claim and should have a relatively long history demonstrated by documentation.

2. Policy and Documentation

From a regulatory viewpoint, clearly defined and stated management policy and a well-managed and complete documentation activity are the only ways the NRC can obtain reliable visibility into other design factors. The organizational record should show a number of years of successful practice under stable policy.

3. Configuration Management

Configuration management is crucial and is absolutely necessary to have confidence that the correct product is built, and that change occurs in an orderly way. Configuration errors are among the simplest and also the most prevalent made in the software industry. Adequate configuration management can be demonstrated by review of past and current company practices. Three configuration management functions of particular note should always be present: change control, interface documentation and control, and delivered product configuration control.

4. Personnel Qualifications

Although this will be a controversial subject, personnel quality continues to be the single most important factor in the designing and coding of a software product. The controversy is not whether intellectual capability is a good thing, but rather who should determine it and how it should be determined. A distinction should be made between managerial ability and technical ability. They are different and often do not occur simultaneously in the same person. Review of personnel qualifications is still a subject of NRC inquiry, but is so important that it cannot be omitted from this list.

5. Independence

V&V, testing, configuration management, and product certification should at least be independent of the managers and programmers responsible for developing the product, and V&V preferably should be done by a different company entirely. Independence can be demonstrated by review of a company's management structure and reward system.

6. Life Cycle

The organization should have a very clear picture of, and a formal model for, the life cycle of its products. This should be clearly reflected as the temporal glue that binds all development and certification activities into an orderly sequence. The use of a life cycle model will be clear from review of development plans.

7. Process Improvement

Management should have a clear picture of the development process and should be prosecuting continuous improvement efforts. This should be demonstrated by a reasonably long (several years)

documented history of this activity. Other checkpoints are development and use of documented internal or external (e.g., national) standards, and use of process models.

8. Process Measurement

Management should be measuring the results of the software process and management's own performance. Without a measurement record, claims of process improvement cannot be substantiated. The database of measurement results is evidence of process and product measurement.

9. Reliability Practices

For ultra-reliable safety software, requirements, development techniques, V&V rigor, and product factors should be guided by the results of hazards and risk analyses. The consistent use of reliability practices can be seen in the documentation of analyses used for planning development, V&V, and designs for prior safety-related products.

10. Requirements

The software development process should produce clear, stable, and validated requirements. Company practices, plans, and example requirements from prior safety-related products provide evidence that clear, stable, and validated requirements are the norm. Positive findings include documented requirements analyses, requirements stability control using configuration management, and the use of prototyping or simulation to understand the implications of requirements more fully.

11. Development Attributes

The software development process should produce clear, unambiguous, documented designs that are traceable item-by-item to requirements. There should be a systematic process for producing software products that are traceable item-by-item to designs. This is demonstrated by documented software development process models, which describe the systematic procedures and attributes of the software development process. The models may be validated by process measurement, as noted in item 8 above.

12. V&V

V&V activities should independently confirm requirements and development attributes and individual product quality. V&V leaves a significant trail of documentation, which can be inspected for prior safety-related projects. Significant positive findings include multi-level testing (e.g., unit, subsystem, and system), and products that are designed to facilitate V&V.

13. Resources

The level of resources and training assigned by management should be appropriate for the difficulty of the software tasks. Resource allocation is documented by management plans and histories of plan execution. Training is documented by personnel assignment records.

14. Development Risks

Management's ability to assess development risk and history of on-time, on-budget, within-specification deliveries is a significant indicator of probable quality. Managerial performance is documented by plans and results for previous safety-related projects similar in scope and nature to current work or future work under the purview of the NRC.

15. Early Problem Detection

The practice of early problem detection and resolution is a positive indicator of eventual product quality and can be demonstrated by documenting detected software errors systematically.

16. Defect Tracking

Defect tracking, root-cause determination, and correction of both the product and the process are positive indicators of process improvement, if the defect-tracking activity is done with due regard for statistical validity. Documentation of this activity provides a record of organizational performance.

B4.2 Other Design Factors

1. Independent Review

Independent review of a software organization, such as ISO 9000 certification, is useful but probably not sufficient for NRC purposes without employing additional criteria.

2. Team Coordination

The design of reactor protection software is not a large software task and should not require large teams to accomplish. If it does, this may be a negative factor because the design may be larger and more complex than necessary. Without explicit management efforts, team coordination may still occur informally, but will not be documented.

3. Product Delivery

The NRC's current requirements for QA inspection (10CFR50) or ITAAC (10CFR52 Part B) are probably sufficient if interpreted to require validation in-plant.

4. Software Maintenance

Software maintenance is not currently an NRC concern for design certification or new-plant delivery. The planning and provision of appropriate software maintenance probably should be.

5. Tools

The use of tools during the development process is considered a positive factor, but the extent and type of usage varies so widely in the industry that no ranking or level of importance useful to the NRC can be ascribed. The software tools market is also so volatile that regulatory requirements tied to particular products would be out of date within six months.

6. COTS

Certification of COTS software, including as used or provided by software developers under scrutiny by the NRC, will be considered in an upcoming report.

7. Contracts

The contractual requirements of ISO 9000 are irrelevant to NRC purposes. Subcontractor requirements of the SEI approach may be necessary, but insufficient, since to depend upon a prime contractor (reactor vendor) to oversee a subcontractor's software development process may be an abdication of NRC responsibility.

8. Miscellaneous Opinions

The miscellaneous opinions offered by the developers surveyed in Task 2 are irrelevant to NRC purposes because they are situational and have no general applicability.

B4.3 Product Factors

Product factors for ultra-reliable reactor safety software are listed below. When present, they indicate that extremely simple design practices are being used, which is a positive factor for ultra-reliable software. Product factors can be ranked in terms of increasing complexity and risk, but the ranking scheme can be circumvented by combining riskier design practices in innovative ways. An unfortunate drawback to product factors is that reviewers reasonably skilled in software development are required to do product review. Product factors include:

- Simple loop
- No interrupts
- Deterministic, predictable timing
- Strong data typing
- No pointers
- No multi-tasking.

B4.4 Negative Factors

The negative factors listed in this group would provide a consistency check against other claims and information provided by a developer seeking NRC approval. Most of these factors have been reported in the general computer science literature as distinguishing software project failures. For instance, requirements instability, schedule-driven development, and repeated failures to achieve predicted cost, schedule, and product quality are hallmarks of several large and costly failures. Negative factors should be regarded as cause for much more intense scrutiny. For convenience, those factors are repeated below:

- There is high turnover.
- Projects are schedule-driven, rather than quality-driven.
- Organizational process history is short or lacking.
- Management cannot enforce stable requirements.
- Management's estimates of product reliability greatly exceed what is actually measurable or provable.
- Management has a record of failing to meet predicted cost, schedule, and quality goals for products.
- The organization fails to track errors and causes.
- The development effort is underfunded.
- The organization exhibits "kill the messenger" syndrome.

Annex A—Design Factors

A.1.0 Part 1—Detailed Design Factors

Design factors are organized under nine headings and are described in one-paragraph appraisals below. The rationale for each heading is described in a single paragraph following the heading. Each design factor description gives the justification for the design factor and notes restrictions where appropriate. The design factors listed are taken from sources and are not the work of the authors. The arrangement, headings, and descriptions represent the opinions or work of the authors.

A.1.1 General Factors

These are factors that apply generally to all members of an organization in all phases of a software development cycle.

A.1.1.1 All levels of the organization are committed to quality.

Since software quality is, like a chain, as strong only as the weakest link, all members of a software development organization must be committed to making quality happen. Management commitment is particularly important because management controls the resources allocated to quality assurance activities.

A.1.1.2 There is longevity in personnel, policy, and process.

The process of building a quality software development organization takes time, by some accounts two years for each incremental improvement in SEI maturity level (Paulk 1993). Therefore, personnel, policies, and the development process must exist and be under improvement for a relatively long period of time. Data on product performance and software process must also be collected for significant periods of time to be statistically valid.

A.1.1.3 Configuration management is used extensively.

Configuration management was cited by all respondents as being crucial to any scheme of product or process control, irrespective of any other software method or process model. Without effective configuration management, it is impossible to determine what has been delivered, how it was produced, who made it, and whether it met requirements.

A.1.1.4 Testing, V&V, and SQA are independent.

Independence of testing, validation and verification, and quality assurance activities from development activities that are under schedule and budget pressure is necessary to prevent compromise of quality for expediency.

A.1.1.5 An appropriate life cycle model is used.

The discipline of using a life cycle model is more important than the actual details of the model selected. A life cycle model allows the various related activities of software development and quality assurance to be coordinated in a rational progression.

A.1.1.6 There is continuous process improvement.

No software development process is perfect, and a good process will degrade without continuous attention. Improvement was a general watchword regardless of actual process details.

A.1.1.7 Reviews, walkthroughs, and inspections are used.

Reviews, walkthroughs, and inspections were recommended at all stages of development and for all products, including V&V and quality assurance products. Code inspections and walkthroughs are credited by one respondent with finding 85% of errors prior even to testing.

A.1.1.8 Automation is used where appropriate.

Automation is suggested for all activities that are tedious, repetitive, error-prone, and sufficiently well-defined that automated software tools can be written to accomplish them. This allows human effort to be redirected to areas where the human intellect is superior to machine performance. Automation may also permit enforcement of standards and customs.

A.1.1.9 Vendors, products, and services are certified.

Products and services used in the development process should be certified to the level required to support the product(s) being developed.

A.1.1.10 Software is the company's primary business.

The company, or division responsible for software, should be in the software development business directly, not as a peripheral activity to the company's real business. This ensures that software concerns and software expertise are sufficiently high in the company's business plans that they receive adequate attention and resources.

A.1.1.11 The organization adapts to changing environments.

The computer industry, and particularly the software industry, has undergone rapid change during its entire existence. Software development organizations and their software development process must continue to adapt to this changing environment, both because old methods used in new situations may be inappropriate, and because the tools and equipment available may force the change.

A.1.1.12 The organizational goal is defect-free software.

Even though achieving this goal may be impossible, no safety-critical software developer should aim to have defects. The defect-free goal and the resources devoted to it are evidence of commitment to quality.

A.1.1.13 Quality must be built in; testing cannot find all defects.

It is not possible to "test in quality." Quality must be designed into the product and that fact should be demonstrated by testing, V&V, and quality assurance activities. Quality, in this definition, means adherence to requirements.

A.1.2 Process Control Factors

These are factors that apply specifically to controlling or measuring the software development process.

A.1.2.1 Processes are defined.

The software development process should be defined in detail so that practitioners can judge whether or not they are accomplishing development according to the process model.

A.1.2.2 Process is stabilized by measurement and feedback.

Performance of actual development activities is measured and compared with the defined process model. If discrepancies exist, either development activities are redirected or the model is changed until a stable, well-understood development process is achieved.

A.1.2.3 The number of process variants is reduced by standardization.

An organization should settle on one or a few process models to guide their development activities, depending upon purpose and business. For instance, a spiral life cycle model with repeated prototypes might be used for products whose requirements are inexactly known but whose failure consequences are low. A waterfall life cycle model might be used for products whose requirements are well known, but whose performance requirements are strict and whose failure consequences are severe. Limiting the number of process variants makes sense because scarce resources can be applied more effectively to process improvement.

A.1.2.4 Processes are improved only after they are stabilized.

Hitting a moving target is always more difficult than hitting a stationary target. Development processes should be stabilized and measurements made so that the effect of changes can be determined and adjustments made in additional change efforts. Two developer organizations suggest that changes to process should be made one at a time, allowing time for stabilization and measurement before making additional changes. This is one contributor to the longevity factor (see section A.1.1.2).

A.1.2.5 Data collection and use of data is balanced.

The amount of data collected should be appropriate to the use of it. Collecting data that will not be used (including usage later in historical databases) wastes effort. Making decisions with inadequate data is just guessing. Data should be collected for historical databases as part of building a long-term process or product history, but not to the extent that it overwhelms short-term data usage activity.

A.1.3 Management Factors

These are factors that are primarily the responsibility of management to implement or enforce.

A.1.3.1 The reward structure matches the quality commitment.

If management gives lip service to quality, but rewards for other performance, other performance is what will be achieved.

A.1.3.2 Management uses process models.

While all persons involved in the software development effort benefit from understanding the process models in use, management's ability to control development processes by allocation of resources and effort is greatly enhanced by using process models.

A.1.3.3 There is constant process measurement and improvement effort.

Software development is a perishable process. Quality can only be maintained by constant measurement and improvement effort. Management's responsibility is to see that this effort is expended, even though it does not contribute immediately to a product.

A.1.3.4 Management makes predictions using models.

Control of process is only achieved by predicting what effect proposed actions will have, and modifying actions to have the desired effect. Management should use model predictions as a first cut at determining what the effect of management actions will be. The use of models predictions and subsequent measurement is essentially feedback control, with the model predictions providing a feedforward component. In the language of control systems, the measurement lag would make an extremely sluggish system or an unstable one without the anticipation provided by predictions.

A.1.3.5 Management achieves predicted cost, schedule, and quality goals more often than not.

It is important that management have a track record of planning, allocating resources, and meeting schedules within cost and quality constraints because the first thing to go under schedule and cost pressures is usually quality.

A.1.3.6 Management controls risks by adopting appropriate strategies.

In the commercial software development world, risk is perceived as failing to deliver a minimally acceptable product on time and more or less within budget. Management uses such strategies as "descoping" (delivering less), delivering with bugs, or renegotiating schedules, depending upon contractual provisions (if any) and commercial conditions. Delivering buggy software is a strategy often used by commercial software vendors trying to hit a market window, although it is greatly disliked by customers.

A.1.3.7 Management abandons methods that do not work.

This may seem like an obvious thing to do, but abandoning a work practice is often a serious career risk for a manager because it is viewed as an admission of error. Mature management expects some percentage of methods attempted to perform relatively poorly, and plans to acquire data about method performance with a view to discontinuing those that do not work well (see item A.1.7.9).

A.1.3.8 Management ensures planning, production, and control of documentation.

Accurate and complete documentation is necessary for product maintenance as well as data collection about organizational performance. Documentation is one of the first things to be neglected under stress, and serves as a sensitive indicator of management performance. Documentation also serves as a

record of organizational longevity and history, validating claims of sufficient experience to be considered at one of the higher SEI maturity levels.

A.1.3.9 Management invites external review.

Nobody is objective about themselves.

A.1.3.10 Improvement takes time — an average of two years.

Respondents were unanimous in noting that no quality software development organization can be put together overnight. Empirically, it appears that about two years are required for the average software organization to move up one rank in the SEI maturity scale.

A.1.4 Personnel Factors

These are factors that characterize the personnel involved in the software development process.

A.1.4.1 Programming skill is not enough; some personnel must be skilled in the problem domain.

When software is applied to problems whose solution has a significant non-software component, as would occur in reactor protection systems, aerospace control systems, or the like, programmers who have no knowledge in the application field are prone to make mistakes of ignorance. In the highly specialized space shuttle program, for example, there is close cooperation between engineers and scientists who are cognizant of astronautics and shuttle systems, and programmers.

A.1.4.2 High intellectual ability of staff is crucial to success.

Most writers in the software development field note that the single greatest factor in assuring quality software is staff quality. A distinction should be made between managerial ability and technical ability. Often, an individual may be adept in only one area.

A.1.4.3 Inaccurate interpersonal communications are an obstacle to producing high-reliability software.

The large organizations in the LLNL survey noted that as software teams get larger, efficient inter-team-member communications become more important, and sometimes become a bottleneck. The corollary of this point is that small teams are preferred.

A.1.4.4 Personnel in influential positions should be highly skilled in all aspects of the development of high-reliability software.

Developing high-reliability software is a special skill that is learned, not inherent. Persons with average schooling or experience cannot be expected to do this and should not be in positions where their inexperience can affect the development process.

A.1.5 Development Factors

These are factors specifically related to the software development process.

A.1.5.1 Configuration management, V&V, and SQA are coordinated with development activities.

This reflects the fact that in an orderly software development process, certain products subject to V&V and SQA are available at process milestones. V&V and SQA products are produced from development products and lose their effectiveness if not fed back into the development process in a timely way.

A.1.5.2 Requirements are stable.

One of the major markers of failed software development efforts is unclear and constantly changing requirements. Stable requirements are crucial to success.

A.1.5.3 A requirements analysis is performed.

Having stable requirements is merely the first step. Requirements must be analyzed to understand their implications. Analysis often reveals inconsistencies, unneeded but expensive specifications, or requirements that may be extremely difficult or impossible to fulfill. Analysis is also necessary when converting requirements provided by non-software specialists to requirements suitable for software.

A.1.5.4 A requirements validation is performed if possible.

Requirements validation is the process of returning to the original statement of requirements and examining the detailed, analyzed list in light of the original. In some cases, such as reachability analysis of communication protocols, requirements validation can be automated.

A.1.5.5 Much of the development effort concerns getting the requirements right.

This is true, at least, of experienced software developers who have discovered that it is easier in the long run to do something once right, than several times wrong. Long and detailed scrutiny of requirements is a marker of successful developers.

A.1.5.6 Prototyping or simulation is an important tool.

Prototypes or simulations are useful in three ways in software development (Preckshot 1993). First, they are often used to demonstrate proposed designs to prospective users as an iterative method of refining requirements. Second, they may be used to test an approach to solving a problem in which there are uncertainties, including hardware performance uncertainties. Third, they may be used in the traditional sense of an engineering prototype, to demonstrate or validate performance of a scalable portion of the final system.

A.1.5.7 Critical components are identified early.

This point was emphasized by several respondents and reflects the view that management must identify where to apply the most valuable resources (personnel) early, so that they have time to solve the problems. This is a form of development risk management.

A.1.5.8 Development activities promote early detection of errors.

It is a widely held view in the software industry that it is less costly to fix errors early in the development process. It is true in general that bug fixes of late errors are often limited by earlier design decisions and the small remaining time (schedule) and funds (budget). Early error detection is another form of development risk management.

A.1.5.9 Defect tracking is done uniformly and consistently.

One of the indications of how well a software development organization is doing is the number of software errors being committed. Error or defect tracking is not very easy, however, and represents considerable effort to do in a statistically valid fashion. Invalid methods of error accounting affect both estimates of software reliability and corrective efforts applied to the software development process.

A.1.5.10 Root causes of defects are determined and corrective actions are taken.

Once errors are identified and tracked, the reasons they occurred should be determined and then two corrective actions should be taken: the product should be fixed and the development process should be modified, if appropriate, to reduce the probability of similar errors in the future. This factor is typical of developers that maintain close control of errors and error causes.

A.1.5.11 Testing is done in several levels, viz. unit, subsystem, system.

It has been found that testing at different levels is necessary because of two countervailing effects. As level becomes more complex (toward system) low-level errors may only be exercised rarely, thus reducing the probability of finding them. On the other hand, complex interaction errors may only exist when the entire software system is assembled. Also, inasmuch as early error detection can only be done on software that is ready early, perforce unit and subsystem testing must be done because the system is not yet available. Consequently, testing at multiple levels of system assembly is done by high-reliability software suppliers.

A.1.5.12 V&V is planned early in the life cycle and results are peer-reviewed.

Validation and verification as an afterthought is a marker of an inexperienced or sloppy developer. V&V should be planned early so that sufficient resources can be allocated to accomplish it and so that there is sufficient time to do it correctly. Peer review ensures that V&V results are not reviewed exclusively by those who planned the tests, analyses, and inspections. This helps avoid "expectation blindness," in which the planners may see only the results they expected to get and ignore signs of possible trouble.

A.1.5.13 The product is designed to validatable and verifiable.

Much as an electronic device can have test points built in for test and calibration, software can be designed with a view to making V&V easier. This also means that design documentation and coding style should permit review and easy understanding by others not directly involved in development.

A.1.5.14 A design philosophy suitable for safety-critical software is used.

In general, this means avoiding the use of "risky" practices. Depending upon the challenges the developer must face (e.g., aerospace vehicle flight control is more difficult than reactor protection systems), the most dependable and least complicated way of solving the software problem should be selected.

A.1.5.15 There is extensive reuse of "middleware."

"Middleware" is defined by the respondent that proposed this factor as middle-level subroutines that are general enough to be used by several applications. Such routines are also known as "trusted" routines, and reuse implies that a software developer maintains a library of trusted subroutines that are well documented, extensively tested, and understood by the programming staff. The advantage of reuse is that scarce intellectual resources are freed for application to problems specific to the software job at hand. The disadvantage is that trusted routines may be misapplied because they "almost" fit the

function needed. Estimates in the literature suggest that reuse can save from 20% to 50% of the effort involved in creating software modules from scratch, but claims of greater savings should be viewed with caution unless the new application is very similar to previous applications.

A.1.5.16 Software layers are identified and managed appropriately according to risk.

This factor recommends an hierarchical structure for software and makes a statement about risk that is ambiguous. From a software developer's viewpoint, the riskiest software layers are those upon which the whole product depends, so these must be done before any version of the product can be delivered. From a regulator's viewpoint, the riskiest software is that software that is essential for safety, followed by that software that is important for safety, followed by all other software. The respondent probably meant the first definition of risk.

A.1.5.17 An appropriate level of complexity is defined for the product and practices are followed that control it.

The minimum level of complexity that the product must have is set by the functional complexity of the requirements the product must meet. Many software products have more than the minimum complexity because of implementation practices or because the designers choose an approach that is unnecessarily complex. Complexity control must occur over the entire life cycle of the product because unneeded complexity can creep in during requirements analysis, design, implementation, or maintenance.

A.1.5.18 Project teams are small (6–8 members).

This factor addresses the difficulty of communication in large project teams. It can be done, but it is difficult to maintain currency and direction in large project teams, and management's role in defining and maintaining critical interfaces and project team communications becomes more important as the team gets larger. Small teams avoid many of the pitfalls.

A.1.5.19 Software interfaces are documented and controlled.

Software interfaces (subroutine calling conventions, system call conventions, interrupt handling, network protocols, distributed system interactions) are always important and form a significant part of design documentation. With large project teams or multiple software contractors, they are even more important. Lack of interface documentation and control is almost always a sign of trouble.

A.1.5.20 Automated tools are used to enforce standards.

Automated tools, if easy to use, are a way of getting everyone on a team to do things the same way (the tool way), and thus provide a relatively low-conflict way of enforcing standards. From a regulator's viewpoint, the use of automated tools improves consistency and performance (the job is more likely to be done), both of which are positive factors.

A.1.6 Reliability and Safety Factors

These are factors directly related to producing safety-critical software.

A.1.6.1 Use of hazards analyses must be part of the development process for safety-critical products.

Hazards analysis shows pathways a system can follow to get into hazardous conditions, and is recommended by several experts to ensure that software takes these pathways into account. Hazards

can also be introduced by the selection of design approaches, certain hardware, software tools, or the use of software itself as a solution to a safety problem. The same expert recommends additional hazards analyses at points during the development life cycle to ensure that existing hazards continue to be covered and that new hazards are not introduced.

A.1.6.2 Diversity used to improve reliability is a system issue; safety is a system issue.

Software diversity (e.g., N-version programming) has not yet been demonstrated to be adequate to counter common-mode failure due to programming error. For this reason, diversity at the system level (i.e., diverse, non-software methods) should be considered for improving total system reliability. Safety is a system issue, not solely a software issue. In safety systems containing software, software is only one of several components that must function correctly to perform the safety functions. Like diversity for reliability, non-software elements should be used to improve total system safety.

A.1.6.3 Accidents are often caused by non-technological factors.

The safety system of which software is a part may be circumvented, turned off, or driven into failure by operator actions. Neither the system or the software should be expected to prevent these problems.

A.1.6.4 Ultra-high reliability (10^{-7} to 10^{-9} failures per demand) cannot be assured by currently known means.

No known method of testing can be or has been applied for sufficient time or number of demands to demonstrate ultra-high reliability. No method of logical analysis has yet been accepted by safety system experts as sufficient to prove that ultra-high reliability has been achieved. Therefore, claims of ultra-high reliability should be viewed with caution. System designs that depend upon ultra-high reliability of components should receive the highest level of scrutiny.

A.1.6.5 The current practical limitation of testing is about 10^{-4} to 10^{-5} failures per demand.

This is based upon approximately two years of testing time on an unchanged product without error. Since software products, including compilers, linkers, and other software tools used to develop safety software, often have a new version cycle of approximately two years, two years may be the practical maximum testing time available in the current commercial environment.

A.1.6.6 Complexity measures are of very limited utility in estimating software reliability or remaining software errors.

The most effective use of complexity metrics is as a guide for allocating resources during development (Preckshot 1993). No complexity metrics have been validated against reliability measures or software errors. In the opinion of software safety experts, complexity metrics are “snake oil” (Lawrence 1992).

A.1.6.7 Degrees of reliability better than 10^{-3} failures per demand require much larger investments.

Estimates of the development cost of space shuttle flight software are that it cost five to ten times what comparable ground support software cost to develop. Additional quality control measures are expensive, and justified where consequences of failure are great.

A.1.7 Negative Factors

These are factors whose presence should be cause for caution or more thorough scrutiny.

A.1.7.1 There is high turnover.

The most obvious implication of high turnover is that building a team of high-quality people with a team memory is impossible. Less obvious is the fact that high turnover is a comment by programmers and managers who leave on the competence of management that is left behind. It should not be ignored.

A.1.7.2 Projects are schedule-driven, rather than quality-driven.

The first victims of a missed deadline are usually quality assurance and documentation. The next victim is the testing program. A "deliver at all costs" mentality is cause for caution.

A.1.7.3 Organizational process history is short or lacking.

Most respondents were explicit about the length of time it takes to build a quality software operation. SEI generally regards maturity level changes as requiring significant time (at least upwards). ISO requires several years to achieve certification.

A.1.7.4 Management cannot enforce stable requirements.

Stable and complete requirements are necessary for quality software products, but the role of management in ensuring this cannot be emphasized enough. Not only must management demand that requirements be locked down, but management itself must not be the source of requirements thrashing. Requirements instability and weak management control are indicators of potential failures.

A.1.7.5 Management's estimates of product reliability greatly exceed what is actually measurable or provable.

Unrealistic claims of product reliability may be an indication that management does not understand the limits of the current state of the art in software development. Such claims should be investigated and management should be given an opportunity to prove its claims.

A.1.7.6 Management has a record of failing to meet predicted cost, schedule, and quality goals for products.

This is typically an indication of management by chaos or paradoxically, schedule-driven rather than quality-driven development. Schedule- and budget-driven development schemes often fail to meet delivery schedules because of product non-performance problems. Something is delivered, but it is not the contracted-for product. A record of failing to meet cost, schedule, and quality goals should be taken seriously as an indicator of deeper troubles.

A.1.7.7 The organization fails to track errors and causes.

An organization's record of errors, causes, and corrective actions is its win-loss track record. No record, or a haphazard record, should be taken in default as meaning a bad record.

A.1.7.8 The development effort is underfunded.

Several of the developers interviewed suggested that most large government software contracts are underfunded by at least a factor of two with the expectation that more funds can be obtained later by litigation, contract expansion, or cost overrun procedures. Whatever the reason, underfunding results in staff transients and failures to carry out "non-essential" activities such as quality assurance,

documentation, and V&V. While it may be difficult for an outside reviewer to estimate what a correct funding profile should be, this negative factor is a very real one.

A.1.7.9 The organization exhibits "kill the messenger" syndrome.

Several of the developers had administrative procedures by which bearers of bad tidings could unburden themselves without jeopardizing their careers. They noted that organizations without these mechanisms were often the last to know about internal problems.

A.1.8 Product Factors

Product factors characterize the software product itself. The factors listed below represent product characteristics that are considered low-risk implementation methods for ultra-reliable software. The presence of these factors is considered a positive indication of lowered complexity or easier error detection.

Product factors are not usually found in standards or process models, because standards setters and process model makers consider that such factors restrict the generality of the standard or process model. Nonetheless, where safety-critical software is concerned, these design factors are useful as product quality indicators. The British Ministry of Defence (MoD) has taken this approach, for instance, in the first draft of its proposed standards for safety-critical software (MoD 1991a, 1991b).

A.1.8.1 No interrupts.

The use of interrupts, beyond a simple clock interrupt, is considered a higher-risk implementation method because of the extra care required to ensure correct synchronization between interrupt code and interrupted code, and to ensure that interrupted code is correctly resumed.

A.1.8.2 No multi-tasking.

Multi-tasking requires context switching and task management in addition to the complications attendant upon using interrupts.

A.1.8.3 Simple loop.

A single-loop program structure is the simplest program organization capable of continuous operation that is possible.

A.1.8.4 Deterministic, predictable timing.

Evidence that software product timing is a predictable function of load, and that load is limited by design is a positive factor. See Preckshot (1993) for additional information.

A.1.8.5 No pointers.

The use of explicit pointers (addresses) of data has been taken by some as a risky practice. The potential exists for errors in programmer-directed address arithmetic which would not exist if named variables were used and the addresses were computed automatically by compiler.

A.1.8.6 Strong data typing.

Data typing permits compilers to detect data misuse errors (e.g., using an integer as if it were a floating point number). This class of error represents a significant proportion of all errors made, and strong data typing with good compilers almost eliminates it.

A.2.0 Part 2—Managerial/Technical View

As required by the statement of work, the design factors are separated into two groups, those primarily involving management considerations, and those involving primarily technical considerations.

A.2.1 Management Design Factors

A.2.1.1 General

There is an organizational commitment to quality.

There is longevity in personnel, policy and process.

The company has experience with safety-critical applications.

There is continuous process improvement.

Automation is used where appropriate.

Vendors, products and services are certified.

Software is the company's primary business.

The organization adapts to changing environments.

The organization's goal is defect-free software.

A.2.1.2 Process Control

Processes are well-defined.

A process is stabilized by measurement and feedback.

The number of process variants is reduced by standardization.

Processes are improved only after they are stabilized.

Data collection and use of data is balanced.

A.2.1.3 Management Action

The company reward structure matches the quality commitment.

Management uses process models.

There is constant process measurement and improvement effort.

Management makes predictions using models.

Management achieves predicted cost, schedule, and quality goals more often than not.

Management controls risks by adopting appropriate strategies.

Management abandons methods that do not work.

Management ensures planning, production and control of documentation.

Management invites external review.

Improvement takes time—an average of two years.

A.2.1.4 Personnel

Programming skill is not enough; some people must be skilled in the problem domain.

High intellectual ability of staff is crucial to success.

Inaccurate interpersonal communications are an obstacle to producing high-reliability software.

Personnel in influential positions should be highly skilled in all aspects of the development of high-reliability software.

Project teams are small.

A.2.2 Technical Design Factors

A.2.2.1 General

Reviews, walkthroughs, inspections are used.

Automation is used where appropriate.

Automated tools are used to enforce standards.

Degrees of reliability better than 10^{-3} failures per demand require large investments.

Configuration management is used extensively and is independent of the development organization.

A.2.2.2 Technical Planning

An appropriate life cycle model is selected.

Development activities promote early detection of errors.

Software layers are identified and managed appropriately according to risk.

A.2.2.3 Requirements Specification

Requirements are stable.

Requirements analysis is performed.

Requirements validation is performed.

Prototyping and simulation are important tools.

A.2.2.4 Design Specification

Critical components are identified early.

A design philosophy suitable for safety-critical software is used.

An appropriate level of complexity is defined for the product and practices are followed that control it.

The product is designed to be verifiable and validatable.

A.2.2.5 Implementation (and unit test)

There is extensive reuse of “middleware.”

Software interfaces are documented and controlled.

Testing is done in several levels, viz. unit, subsystem, system.

A.2.2.6 Integration

Testing is done in several levels, viz. unit, subsystem, system.

A.2.2.7 Installation

Testing is done in several levels, viz. unit, subsystem, system.

A.2.2.8 Software QA

Software QA is independent of development organization.

Quality must be built-in—testing cannot find all defects.

Defect tracking is done uniformly and consistently.

Root cause of defects are determined and corrective actions are taken.

A.2.2.9 Safety

Use of hazards analyses must be part of the development process.

Diversity is used to improve reliability; this is a system issue.

Accidents are often caused by non-technological factors.

Ultra-high reliability cannot be assured by currently known means.

Complexity measures are of very limited utility in estimating software reliability or remaining errors.

A.2.2.10 V&V

V&V is independent of development organization.

Requirements validation is performed.

V&V is planned early in the life cycle, and results are peer-reviewed.

The product is designed to be verifiable and validatable.

A.2.2.11 Testing

Testing is independent of the development organization.

Testing is done in several levels—unit, subsystem, system.

The current practical limitation of testing is about 10^{-4} to 10^{-5} failures per demand.

A.2.2.12 Product Factors

No interrupts.

No multi-tasking.

Simple loop.

Deterministic, predictable timing.

No pointers.

Strong data typing.

A.3.0 Part 3—SEI/ISO Factor List

The SEI/ISO factor list from Ploof & Preckshot (1993) is repeated here for the reader's convenience. Individual factors are lettered rather than numbered to distinguish them from Task 2 factors.

A.3.1 Management (MN)

A. All levels of management are committed to quality.

B. Policies are in place for software development and for assuring software quality.

C. There is a well-defined organizational structure with clear assignments of responsibilities and independence between critical quality activities and software developers.

D. Needed resources are provided.

E. Appropriate training is provided to personnel.

F. Intergroup coordination is planned and executed.

G. There are documented plans for each project, including:
   1. An overall development plan,
   2. A quality management plan,
   3. Software development phase plans,
   4. A product review and corrective action plan.

H. Management is involved in continuous process improvement:
   1. Goals are established and process is measured.
   2. There are independent process audits and management reviews.
   3. Improvement actions are taken.

A.3.2 Software Life Cycle (SL)

I. An organization-wide life cycle model exists.

J. A product life cycle model is created from the organization life cycle model for each product or group of products.

K. The product development plan for each life cycle phase includes:
   1. Progress control,
   2. Validation and verification (V&V) for each phase,
   3. Documentation for each phase,
   4. Planned reviews,
   5. Configuration control,
   6. Quality activities to ensure all of the above.

L. Requirements are clearly specified, understood, and verified.

M. Design is clearly specified, understood, and verified.

N. Software is developed in a defined and disciplined manner.

O. Software is independently tested and validated.

P. Methods, techniques, and tools are integrated into the software development process.

Q. Processes for replication, delivery, and installation are defined and documented.

R. Product maintenance process follows a similar rigorous model.

A.3.3 Supporting Activities (SA)

S. Configuration management is defined and performed.

T. Required documentation is defined and it is produced on time.

U. Measurement of process and product is defined and executed.

V. Commercial off-the-shelf (COTS) software and software tools used during development are of sufficient quality that they do not degrade the quality of the end product.

A.3.4 Contractual (CT)

W. ISO 9000 includes contractual requirements that are not strictly necessary for quality.

X. *(added) SEI has prime contractor/subcontractor contract management goals that are related to DOD interests.*

Annex B: Task 16A Vendor Assessment— Merged List of Factors

This appendix arranges design factors from Task 2 and those discovered in the first work item of Task 16 into related groups. Large **X**'s denote explicit statements or agreement with a particular design factor. Small **x**'s denote implicit agreement that is inferred from similar requirements or concentration on related subjects. A blank means that there is insufficient information to state that agreement exists or that there is specific disagreement. The **X**'s and **x**'s should be interpreted as answers to the question "would the organization agree with this design factor?"

Legend:

X Explicitly required

x Implicitly required or deduced from other requirements

blank Insufficient information.

FACTORS		SEI	ISO	T2	Other
Quality Commitment Group:					
A.	All levels of management are committed to quality.	**X**	**X**	**X**	
A.1.1.1.	All levels of the organization are committed to quality.			**X**	
A.1.1.12.	The organizational goal is defect-free software.			**X**	
A.1.1.13.	Quality must be built in; testing cannot find all defects.			**X**	
A.1.3.1.	The reward structure matches the quality commitment.			**X**	
Policy and Documentation Group:					
B.	Policies are in place for software development and for ensuring software quality.	**X**	**X**	**x**	
C.	There is a well-defined organizational structure with clear assignments of responsibilities and independence between critical quality activities and software developers.	**X**	**X**		
G.	There are documented plans for each project, including:	**X**	**X**		
	1. An overall development plan.				
	2. A quality management plan.				
T.	Required documentation is defined and produced on time.	**X**	**X**		
A.1.3.8.	Management ensures planning, production, and control of documentation.			**X**	
A.1.5.19.	Software interfaces are documented and controlled.	**X**	**X**	**X**	
A.1.1.2.	There is longevity in personnel, policy, and process.	**x**		**X**	

Personnel Qualifications Group:					
D.	Needed resources are provided (i.e., qualified personnel).	**X**	**X**	**x**	
E.	Appropriate training is provided to personnel.	**X**	**X**	**x**	
A.1.4.1.	Programming skill is not enough; some personnel must be skilled in the problem domain.			**X**	
A.1.4.2.	High intellectual ability of staff is critical to success.			**X**	
A.1.4.3.	Inaccurate interpersonal communications are an obstacle to producing high-reliability software.			**X**	
A.1.4.4.	Personnel in influential positions should be highly skilled in all aspects of the development of high-reliability software.			**X**	
A.1.1.2.	There is longevity in personnel, policy, and process.			**X**	
Independence Group:					
O.	Software is independently tested and validated.	**X**	**X**	**X**	
A.1.1.4.	Testing, V&V, and SQA are independent.	**X**	**X**	**X**	
Independent Review Group:					
A.1.3.9.	Management invites external review.			**X**	
A.1.5.12.	V&V is planned early in the life cycle and results are peer-reviewed.			**X**	
H2.	There are independent process audits and management reviews.	**X**			
Resource Group:					
D.	Needed resources are provided.	**X**	**X**	**x**	
E.	Appropriate training is provided to personnel.	**X**	**X**	**x**	
Team Coordination Group:					
F.	Intergroup coordination is planned and executed.	**X**	**x**	**x**	
A.1.5.18.	Project teams are kept small (6–8 members).			**X**	
A.1.4.3.	Inaccurate interpersonal communications are an obstacle to producing high-reliability software.			**X**	
Process Improvement Group:					
H.	Management is involved in continuous process improvement:				
	1. Goals are established and process is measured,	**X**	**X**	**x**	
	2. There are independent process audits and management reviews,	**X**			
	3. Improvement actions are taken.	**X**	**X**	**x**	
A.1.1.2	There is longevity in personnel, policy, and process.	**X**		**X**	
A.1.1.6	There is continuous process improvement.				
A.1.3.2.	Management uses process models.			**X**	
A.1.3.3.	There is constant process measurement and improvement effort.	**x**	**x**	**X**	
A.1.3.7.	Management abandons methods that do not work.			**X**	
A.1.3.10.	Improvement takes time—an average of 2 years.	**X**		**X**	

Process Measurement Group:					
U.	Measurement of process and product is defined and executed.	**X**	**X**	**X**	
A.1.2.1.	Processes are defined.			**X**	
A.1.2.2.	Process is stabilized by measurement and feedback.			**X**	
A.1.2.3.	The number of process variants is reduced by standardization.			**X**	
A.1.2.4.	Processes are improved only after they are stabilized.			**X**	
A.1.2.5	Data collection and use of data is balanced.			**X**	
Development Risk Group:					
G1.	An overall development plan exists for each project.	**X**	**X**		
G4.	A product review and correction plan exists for each project.	**X**	**X**		
K1.	The product development plan includes progress control.		**X**		
A.1.3.4.	Management makes predictions using models.			**X**	
1.3.5.	Management achieves predicted cost, schedule, and quality goals more often than not.	**x**	**x**	**X**	
A.1.3.6.	Management controls risks by adopting appropriate strategies.			**X**	
A.1.5.16.	Software layers are identified and managed appropriately according to risk.			**X**	
Life Cycle Group:					
I.	An organization-wide life cycle model exists.	**X**			
J.	A product life cycle model is created from the organization life cycle model for each product or group of products.	**X**	**x**	**x**	
G3.	(There are documented) software development phase plans (for each project).	**X**	**X**		
K.	The product development plan for each life cycle phase includes:	**X**	**X**		
	1. Progress control,	**X**	**X**		
	2. Validation and verification (V&V) for each phase,	**X**	**X**		
	3. Documentation for each phase,	**X**	**X**		
	4. Planned reviews,	**X**	**X**		
	5. Configuration control,	**X**	**X**	**x**	
	6. Quality activities to ensure all of the above.	**X**	**X**		
A.1.1.5.	An appropriate life cycle model is used.	**x**		**X**	
A.1.5.1.	V&V and SQA are coordinated with development activities.	**x**	**x**	**X**	
Requirements Group:					
L.	Requirements are clearly specified, understood, and verified.	**X**	**X**	**X**	
A.1.5.2.	Requirements are stable.			**X**	
A.1.5.3.	Requirements analysis is performed.			**X**	
A.1.5.4.	Requirements validation is performed, if possible.			**X**	
A.1.5.5.	Much of development effort concerns getting the requirements right.			**X**	

Development Attributes Group:					
L.	Requirements are clearly specified, understood, and verified.	**X**	**X**	**X**	
M.	Design is clearly specified, understood, and verified.	**X**	**X**	**X**	
N.	Software is developed in a defined and disciplined manner.	**X**	**X**	**X**	
V&V Group:					
O.	Software is independently tested and validated.	**X**	**X**		
A.1.5.1	V&V and SQA are coordinated with development activities.	**x**	**x**	**X**	
A.1.5.11.	Testing is done in several levels, viz., unit, subsystem, system.			**X**	
A.1.5.12.	V&V is planned early in the life cycle and results are peer-reviewed.			**X**	
A.1.5.13.	The product is designed to be validatable and verifiable.			**X**	
Product Delivery Group:					
Q.	Processes for replication, delivery, and installation are defined and documented.	**?**	**X**		
A.1.1.3.	Configuration management is used extensively (i.e., at delivery).			**X**	
A.1.1.5.	An appropriate life cycle model is used (i.e., delivery is part of the life cycle).			**X**	
Software Maintenance Group:					
R.	Product maintenance process follows a similar rigorous model as development.	**X**	**X**		
Configuration Management Group:					
S.	Configuration management is defined and performed.	**X**	**X**	**X**	
A.1.1.3.	Configuration management is used extensively.	**X**	**X**	**X**	
Tools Group:					
P.	Methods, techniques, and tools are integrated into the software development process.	**X**	**X**	**x**	
A.1.1.7.	Reviews, walkthroughs, and inspections are used.	**X**	**x**	**X**	
A.1.1.8.	Automation is used where appropriate.	**x**	**X**	**X**	
A.1.3.7	Management abandons methods that do not work.			**X**	
A.1.5.6.	Prototyping or simulation is an important tool.			**X**	
A.1.5.20.	Automated tools are used to enforce standards.			**X**	
COTS Group:					
V.	Commercial off-the-shelf (COTS) software and software tools used during development are of sufficient quality that they do not degrade the quality of the end product.	**X**	**X**		
A.1.1.9.	Vendors, products, and services are certified.	**X**	**X**	**X**	

Contracts Group:					
W.	ISO 9000 includes contractual requirements that are not strictly necessary for quality.		**X**		
X.	SEI has prime contractor/subcontractor contract management goals that are related to DOD interests.	**X**			
Miscellaneous Opinions Group:					
A.1.1.10.	Software is the company's primary business.		**x**	**X**	
A.1.1.11.	The organization adapts to changing environments.	**x**		**X**	
A.1.5.15.	There is extensive reuse of "middleware."			**X**	
Early Problem Detection Group:					
A.1.5.7.	Critical components are identified early.			**X**	
A.1.5.8.	Development activities promote early detection of errors.			**X**	
A.1.5.17.	An appropriate level of complexity is defined for the product and practices are followed that control it.			**X**	
Defect Tracking Group:					
G3.	(There is) a product review and corrective action plan (for each project).	**X**	**X**	**x**	
A.1.1.12.	The organizational goal is defect-free software.			**X**	
A.1.5.9.	Defect tracking is done uniformly and consistently.	**x**		**X**	
A.1.5.10.	Root causes of defects are determined and corrective actions are taken.	**X**	**X**	**X**	
Reliability Practices Group:					
A.1.5.14.	A design philosophy suitable for safety-critical software is used.			**X**	
A.1.6.1.	Use of hazards analyses must be part of the development process for safety-critical products.			**X**	
A.1.6.2.	Diversity used to improve reliability is a system issue. Safety is a system issue.			**X**	
A.1.6.3.	Accidents are often caused by non-technological factors.			**X**	
A.1.6.4.	Ultra-high reliability (10^{-7} to 10^{-9} failures per demand) cannot be ensured by currently known means.			**X**	
A.1.6.5.	The current practical limitation of testing is about 10^{-4} to 10^{-5} failures per demand.			**X**	
A.1.6.6.	Complexity measures are of very limited utility in estimating software reliability or remaining software errors.			**X**	
A.1.6.7.	Degrees of reliability better than 10^{-3} failures per demand require much larger investments.			**X**	

Negative Factors Group:					
A.1.7.1.	There is high turnover.			**X**	
A.1.7.2.	Projects are schedule-driven, rather than quality-driven.		**x**	**X**	
A.1.7.3.	Organizational process history is short or lacking.	**x**		**X**	
A.1.7.4.	Management cannot enforce stable requirements.			**X**	
A.1.7.5.	Management's estimates of product reliability greatly exceed what is actually measurable or provable.			**X**	
A.1.7.6.	Management has a record of failing to meet predicted cost, schedule, and quality goals for products.	**x**	**x**	**X**	
A.1.7.7.	The organization fails to track errors and causes.	**x**	**x**	**X**	
A.1.7.8	The development effort is underfunded.			**X**	
A.1.7.9	The organization exhibits "kill the messenger" syndrome.			**X**	
Product Factors Group:					
A.1.8.1.	No interrupts.				*
A.1.8.2.	No multi-tasking.				*
A.1.8.3.	Simple loop.				*
A.1.8.4.	Deterministic, predictable timing.				**
A.1.8.5.	No pointers.				*
A.1.8.6	Strong data typing				*

[*] See MoD 1991a, 1991b.

[**] See Preckshot 1993.